REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : GLSD09/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Génie logiciel et Systèmes Distribués (GLSD)**

L'utilisation du machine learning pour la détection des erreurs des programmes

---

Par :
ELGUESS IMANE

Soutenu le ../../…. devant le jury composé de :

	grade	Président
Mme Mohammedi Amira	M.A.A	Rapporteur
	grade	Examinateur

Année universitaire 2020-2021

Remerciements

Je remercie avant tout le Bon Dieu de m'avoir donné le savoir et la faculté de pouvoir poursuivre mes études. Mes remerciements les plus sincères á ma directrice de mémoire Mme Mohammedi Amira, Elle qui m'a guidé avec ses orientations, ses conseils et ses critiques tout au long de ce travail de recherche en me laissant la liberté dont j'avais besoins. Je ne peux que lui être reconnaissante surtout pour ses qualité intellectuelles et humaines.

J'adresse mes sincères remerciements aux membres du jury, qui ont accepté d'évaluer mon travail.

Je remercie aussi tous les membres du laboratoire d'informatique de l'université de Biskra pour leur disponibilité et leur précieuse aide, ainsi que l'ensemble du personnel du département d'informaique pour la formation fournie durant les 5 annés passées.

Et également à mes camarades, amis pour leurs aides précieuses.

Dédicaces

JE DÉDIÉ CE MÉMOIRE À :

Ma chère mère Hadia Ardjoune et mon père Rabie

Dont le mérite,les sacrifices et les qualité humaines m'ont permis de vivre ce jour.

Ma soeur et ma deuxîeme mère Djahida

pour tout ses encouragements et son soutien tout au long de mes études.

Ma grande soeur Noura.

Mon frère Mohamed Rachid

Mes petits neveux

Oussama , Adem ,Adem

Tous les gens qui m'aiment Mohammed ,Khalissa,Souria

(Elguess Imane)

Résumé

Le problème de la correction automatique des erreurs de programmation est un sujet de recherche très actif dans le domaine du génie logiciel. Il s'agit d'un problème difficile car la correction d'une seule erreur peut nécessiter l'analyse de tout le programme. La détection automatique des programmes a attiré l'attention en raison de son potentiel de réduction des coûts de débogage. L'objectif de notre travail est d'exploiter le domaine fructueux du machine learning à la détection des erreurs des programmes. Vue l'indisponibilité de datasets adaptée à notre problématique, nous avons utilisé une dataset (JavaScript Identifiers) et nous avons fait l'hypothèse que l'occurrence d'une variable dans une instruction est une erreur. Nous avons utilisé pour l'implémentation de notre solution les réseaux de neaurones.

Abstract

The problem of automatic correction of programming errors is a very active research topic in the field of software engineering. This is a difficult problem because correcting a single error may require scanning the entire program. Automatic program detection has gained attention because of its potential to reduce debugging costs. The objective of our work is to exploit the fruitful field of machine learning for the detection of program errors. Given the unavailability of datasets adapted to our problem, we used a dataset (JavaScript Identifiers) and we made the assumption that the occurrence of a variable in an instruction is an error. We used neural networks for the implementation of our solution.

Table des matières

Table des figures **ii**

1 État de l'art sur :La détection automatique des bogues de programmes **2**

1.1 Introduction 3

1.2 La signification des bogues logiciels 4

1.2.1 Les schémas de classification des bogues logiciels : . . . 4

1.2.2 Les téchniques de classification des bogues logiciels : . . 5

1.3 La détection automatique des bogues de programmes 6

1.3.1 Notions de base : 8

1.4 Traveaux Connexes 12

1.4.1 Repairnator, un robot autonome pour réparer les bogues informatiques : 12

1.4.2 BugFix : 13

1.4.3 Prophet : 14

1.5 Conclusion 16

2 Machine Learning **17**

2.1 introduction 18

2.2 Définition de l'apprentissage automatique 19

2.3 Types de L'apprentissage automatique 19

2.3.1 Apprentissage supervisé 19

2.3.2 Apprentissage non supervisé 20

2.3.3 Apprentissage semi supervisé 21

2.3.4 Apprentissage par renforcement 21

2.4 Les algorithmes du machine learning 22

2.4.1 Les réseaux de neurones artificiels 22

2.4.2 Arbres de décision 24

2.4.3 Machines à vecteurs de support 25

2.4.4 Naïve Bayes 27

2.4.5 La régression logistique 27

2.5 Conclusion 27

3 Conception du système **29**

3.1 Introduction 30

3.2 Architecture générale 30

3.3 Architecture détaillée 31

3.3.1 Entraînement : 32

3.3.2 Validation : 33

3.4 Conclusion 34

4 Implémentation et résultats **35**

4.1 Introduction 36

4.2 Outils utilisées 36

4.2.1 Logiciels utilisés 36

4.2.2 Langages utilisés 37

4.2.3 Bibliothèques utilisées 38

4.2.4 Datasets 43

4.2.5 Résultats Obtenus : 44

4.3 Conclusion 46

Bibliographie **52**

Table des figures

1.1 Processus de guérison et de réparation [9] 10

2.1 Les types de machine learning 22

2.2 La structure de réseau de neurone [12] 23

2.3 Exemple d'un arbre de décision 26

1.1 Processus de guérison et de réparation

2.1 Les types de machine learning 2.2La structure de réseau de neurone.

2.3Exemple d'un arbre de décision

3.2 Architecture générale

3.3.2 L'utilisation des réseaux de neurone dans notre système

4.2.1 Spyder(Python)

2 Une couche caché MLP

4.2.5 Plot de la visualisation de la détection des erreurs.

4.2.4Une partie de notre dataset

4.2.5 dummies(dataframe)matrice

Introduction générale

Le test logiciel est toujours un processus pénible, long, et coûteux. Des études récentes ont montré que les activités de débogage représentent souvent environ 50% des coût de développement. De nombreux facteurs contribuent au coût du débogage, mais le plus important est l'effort manuel considérable qui est encore nécessaire pour identifier et éliminer les défauts.

Malgré les nombreuses ressources consacrées au développement et au test des programmes, les programmes d'aujourd'hui, avec leur échelle et leur complexité toujours croissantes ont tendance à contenir des bogues qui sont difficiles à suivre et à corriger, Ce qui rend peu pratique pour les programmeurs de les localiser manuellement. l'automatisation des activités de débogage essentiellement liés à l'identification des instructions qui sont susceptibles d'être défectueuses à l'isolement des entrées ou des états d'application spécifiques ou des états de l'application qui peuvent provoquer des défaillances, et avec la détection d'événements anormaux qui peuvent expliquer partiellement la raison d'une défaillance.

L'apprentissage automatique est un sous domaine de l'intelligence ar-

tificielle qui fait référence au développement, l'analyse et l'implémentation d'algorithme permettant à une machine d'apprendre à partir d'un ensemble de données. Ce processus est inductif,il tente de généraliser les relations extraites de l'ensemble d'apprentissage à tout l'espace de données de la source.

L'objectif de notre travail est d'exploiter le domaine fructueux du machine learning à la détection des erreurs des programmes. Vue l'indisponibilité de datasets adaptée à notre problématique, nous avons utilisé une dataset (JavaScript Identifiers) et nous avons fait l'hypothèse que l'occurrence d'une variable dans une instruction est une erreur. Nous avons utilisé pour l'implémentation de notre solution les réseaux de neaurones.

Notre mémoire se compose de quatre chapitres :

1. État de l'art sur la détection automatique des bugs de programmes ;
2. Machine learning : expliquant son principe et les modèles utilisés pour son implémentation ;
3. Conception du système : décrivant l'architecture générale et l'architecture détaillée de notre application ;
4. Implémantation et résultats : présentant la mise en oeuvre de notre travail et les résultats obtenus ;
5. Nous cloturons le présent manuscrit par une conclusion générale.

Chapitre 1

État de l'art sur :La détection automatique des bogues de programmes

1.1 Introduction

Les tests et le débogage ont toujours été la partie la plus longue du processus de développement logiciel et nécessitent beaucoup de ressources humaines. Après la découverte d'une erreur, le corriger manuellement pour réparer le programme bogué reste une tâche difficile et laborieuse pour les développeurs. C'est pourquoi la technologie de réparation automatique du programme. Les chercheurs ont exploré et proposé diverses méthodes et outils novateurs, rapprochant l'idée de la réalité.

Le but de ce chapitre est de présenter d'une part quelques rappels indispensabes et nécessaires à la compréhension de ce mémoire et d'autre part faire une synthése bibliographique de quelques traveaux connexes liés au domaine de la détection et la correction automatique des programmes . Nous présentons dans un premier temps tout ce qui concerne la correction automatique des programmes en utilisant les techniques du machine learning,ensuite comment les détecter.

1.2 La signification des bogues logiciels

L'année 2017 a été désignée comme l'année où les bugs logiciels ont envahi le monde, Il s'agit d'un rappel pertinent du fait que les ingénieurs en logiciels ne sont que des êtres humains et qu'ils ont leurs faiblesses lorsqu'il s'agit de produire des logiciels sans bogues. Les bogues sont signalés dans un système de suivi des bogues, validés par une équipe de triage, attribués à quelqu'un pour les corriger, et finalement vérifiés et fermés [6].

Définitions et Concepts :

la définition de "défaut" par la norme IEEE 1044 (2009) : Une imperfection ou une déficience d'un produit logiciel lorsque ce dernier ne répond pas à ses exigences ou à ses spécifications, telles qu'elles sont définies au moment considéré, et qu'il doit être soit réparé, soit remplacé.

Un bogue : Une imperfection ou une déficience d'un produit logiciel lorsque ce dernier ne répond pas à ses exigences ou à ses spćifications, telles que définies au moment de l'achat. logiciel ne répond pas à ses exigences ou à ses spécifications, telles que définies au moment considéré, et doit être soit réparé, soit remplacé [8].

1.2.1 Les schémas de classification des bogues logiciels :

La taxonomie de classification des bogues la plus ancienne et la plus populaire a été proposée par IBM , qui a introduit la classification orthogonale

des défauts (ODC). Cette taxonomie comprend 13 catégories qui permettent aux développeurs de séparer les bogues en fonction de leur impact sur le client, l'accent étant ainsi mis sur l'effet produit par les bogues plutôt que sur leur cause première[7].

Un autre schéma populaire de caractérisation des bugs a été développé par Hewlett-Packard Dans ce cas, les bugs sont caractérisés par trois attributs :

1. Origin : c'est-à-dire l'activité au cours de laquelle le défaut a été introduit (par exemple, pendant la spécification des exigences ou la conception).
2. Mode : qui décrit les scénarios conduisant à un bug
3. Type : qui décrit plus en profondeur l'origine d'un bug, en précisant s'il est lié au matériel ou au logiciel. Il est important de noter que l'attribut "type" de ce schéma de classification n'est pas destiné à être utilisé par les utilisateurs. de ce schéma de classification n'est pas destiné à être utilisé pour préciser la cause profonde d'un bogue (par exemple, un problème de performances), mais il fournit plutôt plus de contexte sur l'emplacement d'un bogue [7].

1.2.2 Les téchniques de classification des bogues logiciels :

Outre les schémas de classification, un certain nombre de travaux antérieurs ont concu des approches automatisées pour classer les rapports de bogues. Antoniol et al ont utilisé un modèle d'apprentissage automatique

pour distinguer les bogues des demandes de nouvelles fonctionnalités dans les rapports de bogues, avec une précision de 77% et un rappel de 82% Dans notre cas, nous ne considérons que les rapports de bogues signalant effectivement des problémes des applications considérées, puisque notre objectif est de classer les bogues. Hernandez-Gonzalez et al ont proposé une approche pour classer l'impact des bogues selon la taxonomie ODC : l'étude empirique menée sur deux systémes, Compendium et Mozilla, a montré de bons résultats [7]. Dans le même temps, Huang et al, en se basant sur la classification des ODC, ont proposé Auto ODC, une approche visant à automatiser la classification des ODC. en la considérant comme un problème de classification de texte supervisé et en intégrant l'expérience et la connaissance du domaine des experts en ODC, ils ont construit deux modèles formés avec deux classificateurs différents tels que Naive Bayes et Support Vector Machine sur une liste de défauts plus importante extraite de FileZilla. une liste de défauts plus importante extraite de FileZilla ,Ils ont rapporté des résultats prometteurs[7].

1.3 La détection automatique des bogues de programmes

L'analyse statique de programme a pour but de déterminer si un programme réponds à certaines exigences sans exécuter le programme. Comme le programme n'est pas exécuté, l'analyseur statique crée une abstraction du programme sur laquelle les exigences sont évaluées. l'automatisation des activités de débogage portant essentiellement sur l'identification des états sus-

ceptibles d'être défectueux, sur l'isolement des entrées spécifiques ou des états de l'application susceptibles de provoquer des défaillances et sur la détection des événements anormaux qui peuvent expliquer partiellement la raison d'une défaillance[5]. Les techniques de détection d'anomalies peuvent détecter les opérations qui sont exécutées par une application lors de défaillances, mais pas lors d'exécutions correctes. Ces opérations peuvent expliquer pourquoi et comment une application a échoué[5].

Les techniques de détection des anomalies exploitent généralement des approches d'exploration des spécifications pour générer automatiquement des modèles qui représentent le comportement légal d'une application, puis utilisent ces modèles pour analyser les exécutions ratées et déterminer les événements anormaux[5].

Récemment, les chercheurs se sont concentrés sur un nouveau type de méthode, la technologie de réparation de programme. L'idée clé de ces technologies est d'essayer de réparer automatiquement le système logiciel en générant un programme de réparation réel qui peut être vérifié par les testeurs avant d'être finalement accepté, ou il peut être personnalisé pour s'adapter au système. L'avantage d'utiliser ces techniques est que la réparation peut à la fois expliquer la cause de la panne et apporter des solutions possibles au problème [9].

1.3.1 Notions de base :

Deux méthodes bien connues pour gérer automatiquement les échecs de programme sont les technologies de détection et de réparation,Bien que les principes soient similaires, elles sont différentes et utilisent des solutions différentes pour faire face aux échecs.

1. La solution de détection logicielle : "bottom-up approach", détecte et réagit aux pannes logicielles sur site en effectuant les ajustements nécessaires pour restaurer le système en fonctionnement normal. Ces ajustements ne sont pas déployés au niveau du code source, mais sont appliqués à l'application déployée au moment de l'exécution , sur l'application déployée pour prévenir ou atténuer les défaillances.

   Plusieurs défaillances similaires sur la même instance logicielle peuvent déclencher le même processus de réparation plusieurs fois[9].
2. Les solutions de réparation de logiciels :"top-down approach", détectent les défaillances des logiciels, localisent les endroits où un correctif pourrait être appliqué et font le nécessaire des ajustements pour corriger le défaut, et ainsi prévenir d'autres défaillances causées par le même défaut . Les ajustements sont générés en interne, par exemple au moment des tests et de la conception, et déployés au niveau du code source.

Les techniques de détection et de réparation réagissent aux défaillances en exécutant certaines opérations. Nous distinguons les opérations de guérison, qui sont les opérations appliquées au moment de l'exécution pour trans-

former une exécution défaillante en une exécution réussie, et les opérations de réparation, qui sont les opérations effectuées sur le code source du programme pour supprimer la faute qui a causé la défaillance. sur le code source du programme afin de supprimer la faute qui a causé une défaillance [9].

La figure 1.1 illustre graphiquement les activités impliquées dans les processus de guérison et de réparation, représentés respectivement sur les côtés gauche et droit de la figure. Les deux processus commencent d'un programme défaillant qui s'écarte de son comportement prévu, au moins pour certaines exécutions. L'activité de détection des défaillances commune aux deux processus, est chargée de classer les exécutions comme de classer les exécutions comme des échecs ou des exécutions sans échec.

Le processus de guérison illustré sur le côté gauche de la figure 1.1. est composé de deux étapes qui peuvent être exécutées de manière itérative.

L'étape de guérison consiste à exécuter une opération de guérison qui peut empêcher une défaillance qui a été détectée.

L'étape de vérification vérifie si l'application fonctionne comme prévu après l'opération de guérison. Toutes les techniques de guérison n'incluent pas toutes une étape de vérification.

Les opérations de guérison sont appliquées en supposant simplement qu'elles ne peuvent avoir qu'un effet positif ou neutre sur le système. Si l'étape de vérification détecte que l'opération de guérison a échoué, l'étape de guérison peut être répétée, par exemple en essayant une autre opération de guérison

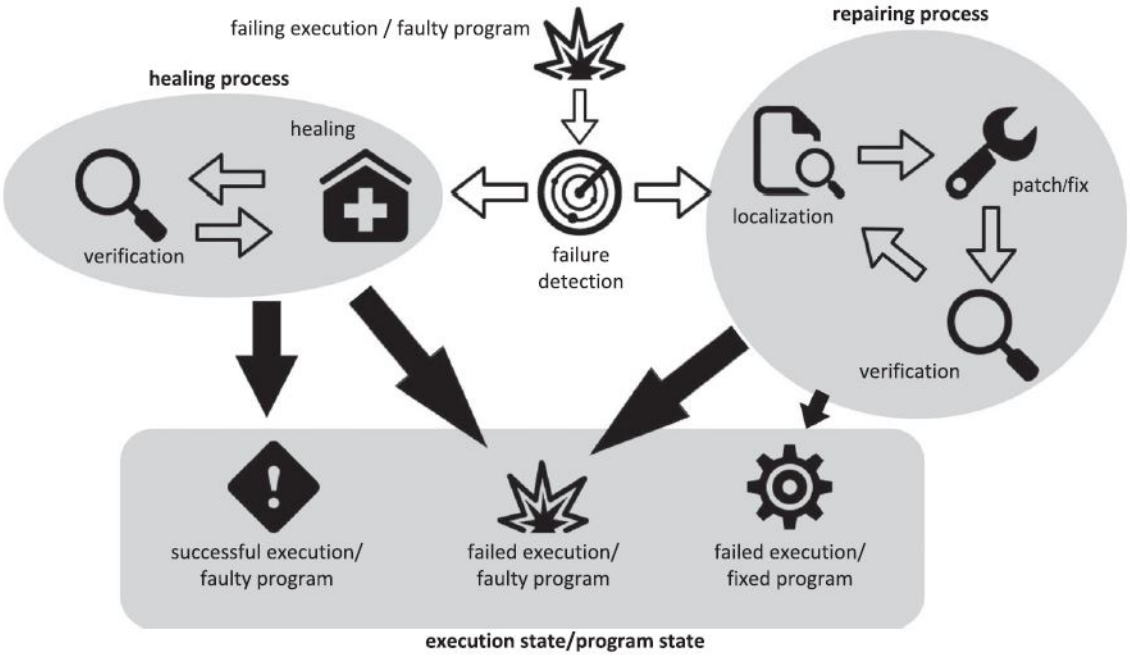

FIGURE 1.1 – Processus de guérison et de réparation [9]

différente et en vérifiant à nouveau le système, jusqu'à ce que la défaillance soit guérie ou qu'aucune autre action ne soit possible.

Le processus de guérison nécessite généralement des opérations et de vérification rapides, car elles sont à la fois Exécuter lorsque l'application est en cours d'exécution sur le site.

Puisque l'objectif premier des techniques de guérison est de transformer des exécutions défaillantes en exécutions réussies,indépendamment de la faute qui est à l'origine du problème, le processus de guérison peut produire deux résultats possibles exécutions réussies/programme défectueux, c'est-à-

dire que les exécutions ont été guéries mais le programme est toujours défectueux.

Le processus de réparation illustré sur le côté droit de la figure 1.1. est composé de trois étapes.

1. L'étape de localisation : identifie les endroits où une correction pourrait être appliquée (notez que les déclarations défectueuses ne sont pas toujours les seuls bons emplacements pour les correctifs).
2. L'étape de patch/fixe : génère des correctifs qui modifient le logiciel dans les emplacements de code renvoyés par l'étape de localisation [11].
3. L'étape de vérification : vérifie si le correctif synthétisé a effectivement réparé le logiciel. Les étapes de correction et de vérification peuvent être itérées plusieurs fois pour de multiples emplacements jusqu'à ce que le défaut ait été corrigé, jusqu'à ce qu'aucun autre correctif ne puisse être généré, ou jusqu'à ce que le temps alloué au processus de réparation soit épuisé [11].

La guérison et la réparation des logiciels peuvent ou non impliquer une intervention humaine. Dans le cas de la réparation et de la guérison automatique, l'homme ne fait que superviser le processus. Alors que,

lorsque l'humain est complétement hors de la boucle, ces techniques sont respectivement appelées auto-guérison logicielle et auto-réparation logicielle.

1.4 Traveaux Connexes

Dans cette section nous allons discuter sur quelques traveaux connexes qui ont le même principe lequel est générer automatiquement de telles suggestions à l'aide d'une approche d'apprentissage automatique qui prend en compte les connaissances acquises à partir des bogues précédents qui ont déjà été identifiés et corrigés.

1.4.1 Repairnator, un robot autonome pour réparer les bogues informatiques :

Repairnator est un robot doté d'une intelligence artificielle, Né de l'équipe projet Spirals*, il a été développé dans le cadre de l'ADT LibRepair obtenue en 2016 par Simon Urli, chercheur postdoctorant. Repairnator est le premier repair bot capable de faire de la réparation automatique de bugs informatiques à grande échelle. En somme, un logiciel efficace et autonome qui sauve des erreurs de code les autres logiciels. Une solution en open source mise à disposition de tous les développeurs [4].

Un logiciel qui répare automatiquement les bugs logiciels : Le responsable de l'équipe-projet Spirals Lionel Seinturier précise : "... Repairnator est un logiciel informatique qui propose de réparer automatiquement les bugs d'autres logiciels. Il s'adresse aux développeurs et fonctionne sur les plates-formes collaboratives d'échange de code comme GitHub, qui a récemment été racheté par Microsoft" [4].

1.4.2 BugFix :

BugFix requiert comme entrée un programme défectueux et une suite de tests correspondante contenant au moins un cas de test défaillant.

Le but de cet outil est de calculer et de rapporter une liste hiérarchisée de suggestions de bogues pour une situation de débogage donnée pour un énoncé de programme suspecté d'être défectueux.

La situation de débogage peut être considérée comme une caractérisation des détails statiques et dynamiques particuliers d'une déclaration suspecte qui est suspect qui est débogué.

Une suggestion de Bug-Fix est une description textuelle de la manière de modifier un énoncé donné de telle sorte qu'un bug dans l'énoncé est susceptible d'être corrigé. Un x réel effectué par un développeur est représenté textuellement par une description de bug-fix. Cette outil est construit sur des concepts de la communauté de l'apprentissage automatique qui permettent à l'outil d'apprendre de nouvelles situations de débogage et les bogues correspondants qui sont rencontrés au fil du temps. Grâce à une utilisation continue, la capacité de l'outil à rapporter des suggestions de bug-fix très pertinentes pour les nouvelles situations de débogage devrait s'améliorer.

Cet objectif est ceci est accompli en maintenant une base de données de scénarios de bug-x décrivant les différentes situations de débogage et les descriptions de bug-fix correspondantes rencontrées précédemment par l'outil. À partir de cette base de données, un algorithme d'apprentissage automatique de règles d'association peut être appliqué pour automatiquement une base de

connaissances de règles, mettant en correspondance situations de débogage (générales et spécifiques) aux descriptions de bogues descriptions de bug-fix correspondantes.

1.4.3 Prophet :

Prophet, un nouveau système de génération de patchs qui fonctionne avec un ensemble de correctifs humains réussis obtenus à partir de pour apprendre un modèle de code correct probabiliste et indépendant de l'application. Il génère un espace de correctifs candidats, utilise le modéle pour classer les correctifs candidats par ordre de candidats par ordre de correction probable, et valide les correctifs classés contre une suite de cas de test pour trouver les correctifs corrects. Les résultats expérimentaux montrent que, sur un ensemble de référence de 69 éfauts réels provenant de huit projets huit projets de logiciels libres, Prophet surpasse de manière significative le système de génération de correctifs à l'état de l'art [11].

Prophet utilise ce modèle pour générer automatiquement des correctifs corrects qui réparent les défauts dans les applications incorrectes :

— Modèle appris de code correct : En travaillant avec un ensemble de correctifs patches réussis obtenus à partir de dépôts de logiciels open-source, Prophet apprend un modèle probabiliste de code correct. Il utilise ce modèle pour classer et identifier les correctifs corrects dans un espace de correctifs automatiquement (dont seulement une petite fraction est correcte).

— Interactions du code : Chaque correctif insère un nouveau code dans le programme. Mais la correction ne dépend pas seulement du nouveau code elle dépend aussi de la façon dont ce nouveau code est utilisé. elle dépend aussi de la façon dont ce nouveau code interagit avec l'application dans laquelle il est inséré. Le modèle de correction appris fonctionne donc avec des caractéristiques qui capturent les aspects critiques de comment le nouveau code interagit avec le code environnant de l'application[11].

Résultats expérimentaux :

Ils ont évaluées Prophet sur 69 défauts du monde réel tirés de huit grandes applications open source.

La figure1.2. résume les résultats pour Prophet, SPR, Kali, GenProg et AE. Il y a une ligne dans le tableau pour chaque application de référence. La première colonne (App) présente le nom de l'application, la deuxième colonne (LoC) présente le nombre de lignes de code dans chaque application, et la troisième colonne (Tests) présente le nombre de cas de test dans la suite de tests pour cette application. La quatrième colonne (défauts/Modifications) contient des entrées de la forme X/Y, oú X est le nombre de défauts exposés dans chaque application et le nombre de cas de tests dans la suite de tests pour cette application. et Y est le nombre de changements de changements de fonctionnalité exposés. L'ensemble de référence contient un total de 69 défauts exposés et 36 changements de fonctionnalité exposés. Les cinquième à neuvième colonnes (Plausible) résument les correctifs plausibles que chaque

système trouve. Chaque entrée est de la forme X/Y,où X est le nombre des 69 défauts pour lesquels le système correspondant trouve au moins un correctif plausible,et Y est le nombre de 36 changements de fonctionnalité pour lesquels chaque système trouve au moins un correctif plausible. Les dixième à quatorzième colonnes (Correct) récapitulent les correctifs corrects que chaque système trouve. Pour Prophet et SPR, chaque entrée est de la forme X,Y/Z. Ici, X est le numéro des 69 défauts pour lesquels Prophet ou SPR trouve un correctif correct avant le délai de 12 heures ,Y est le nombre de défauts pour lesquels Prophète ou SPR trouve un patch correct en tant que premier correctif à valider, et Z est le nombre de changements de fonctionnalité pour lesquels Prophet ou SPR trouve un correctif correct avant le délai de 12 heures.

1.5 Conclusion

Dans ce chapitre, les différents concepts des bogues de programmes et l'automatisation de réparation des bogues logiciels ont été présentés. Nous avons vu leur signification, caractéristiques ainsi que les différents modèles utilisés pour leur détection. L'application du machine learning a particulièrement permis d'apporter des solutions intéressantes à ce problème. Dans le chapitre suivant, nous allons nous concentrer précisément sur les méthodes d'apprentissage automatique et choisir la méthode qui convient le mieux au problème de détection des bugs.

Chapitre 2

Machine Learning

2.1 introduction

Un programme informatique tente de résoudre un problème pour lequel nous avons la solution. Par exemple : calculer la moyenne générale des étudiants, classer les étudiants selon leur moyenne. . .

Pour certains problèmes, nous ne connaissons pas de solution exacte et donc nous ne pouvons pas écrire de programme informatique.

Par exemple : reconnaâitre automatiquement des chiffres écrits à la main à partir d'une image scannée, déterminer automatiquement une typologie des clients d'une banque, jouer automatiquement aux échecs contre un humain ou un autre programme. . .

En revanche, pour ces problèmes il est facile d'avoir une base de données regroupant de nombreuses instances du problème considérée.

L'apprentissage automatique consiste alors à programmer des algorithmes permettant d'apprendre automatiquement de donn´ees et d'expériences passées, un algorithme cherchant à résoudre au mieux un problème considéré [12].

Ce chapitre permet de se familiariser avec les différentes notions liées à l'apprentissage automatique, sous ses formes supervisées et non supervisées, utilisé pour résoudre notre problématique.

2.2 Définition de l'apprentissage automatique

L'apprentissage automatique est un sous domaine de l'intelligence artificielle qui fait référence au développement, l'analyse et l'implémentation d'algorithme permettant à une machine d'apprendre à partir d'un ensemble de données. Ce processus est inductif,il tente de généraliser les relations extraites de l'ensemble d'apprentissage à tout l'espace de données de la source. Plusieurs auteurs ont défini le processus d'apprentissage[1],

Selon Simon :"L'apprentissage automatique désigne l'ensemble des changements dans un système qui lui permettent de réaliser une même tâche,ou des tâches similaires, de manière plus efficace ou plus efficiente au cours du temps" Vincent a défini l'apprentissage automatique comme,"Une tentative de comprendre et de reproduire l'habileté humaine d'apprendre de ses expérience passées et de s'adapter dans les systèmes artificiels"

2.3 Types de L'apprentissage automatique

On distingue différents types d'algorithmes de Machine Learning, qui sont déférents dans leur objectifs et leur fonctionnements :

2.3.1 Apprentissage supervisé

Il s'agit d'une méthode dans laquelle les données d'entrée actuelles sont utilisées pour atteindre l'ensemble des résultats. Il existe deux types d'ap-

prentissage supervisé : l'apprentissage supervisé par classification et l'apprentissage supervisé par régression.

1. Classification : Répartir les données dans les catégories définies sur l'ensemble des données en fonction de leurs caractéristiques spécifiques.
2. Régression : Prédire ou conclure les autres caractéristiques des données sur la base de certaines caractéristiques disponibles.

2.3.2 Apprentissage non supervisé

La différence entre l'apprentissage supervisé et non supervisé est que dans l'apprentissage non supervisé, les données de sortie ne sont pas données. Le processus d'apprentissage se produit en utilisant les relations et les connexions entre les données. De plus, l'apprentissage non supervisé n'a pas de données d'entraînement. Il existe également deux types d'apprentissage non supervisé : le regroupement et l'association [12].

1. Regroupement : Trouver les groupements de données qui sont similaires les uns aux autres lorsque les groupements inhérents aux données ne sont pas connus.
2. Association (Clustering) : Détermination des relations et des connexions entre les données d'un même ensemble de données. Déduction de caractéristiques : Dans certains cas, bien que de nombreuses caractéristiques des données soient connues, les caractéristiques liées au groupe et à la catégorie des données ne peuvent être déterminées.

Dans de tels cas, la sélection d'un sous-groupe de caractéristiques ou l'obtention de nouvelles caractéristiques combinant les caractéristiques est appelée déduction de caractéristiques.

2.3.3 Apprentissage semi supervisé

L'apprentissage supervisé et non supervisé est inadéquat lorsque les données étiquetées sont inférieures aux données non étiquetées. Dans ce cas, les données non étiquetées, qui sont très insuffisantes, sont utilisées pour déduire des informations sur celles-ci. Et cette méthode est appelée apprentissage semi-supervisé.

La différence entre l'apprentissage semi-supervisé et l'apprentissage supervisé réside dans l'ensemble de données étiquetées dans l'apprentissage supervisé, les données étiquetées sont plus nombreuses que les données à prédire. En revanche, dans l'apprentissage semi-supervisé, les données étiquetées sont moins nombreuses que les données à prédire [12]. La figure2.4.3. illustre les types de machine learning et leur utilisation.

2.3.4 Apprentissage par renforcement

Il s'agit d'un type d'apprentissage dans lequel les agents apprennent via un systÃ¨me de récompense.

Bien qu'il y ait un point de départ et un point d'arrivée, le but de l'agent est d'utiliser le chemin le plus court et le plus correct pour atteindre l'objectif. Lorsque l'agent emprunte les bons chemins, il

reÃ§oit des récompenses positives. En revanche, s'il emprunte les mauvais chemins, il reÃ§oit des récompenses négatives. L'apprentissage se produit sur le chemin de l'objectif

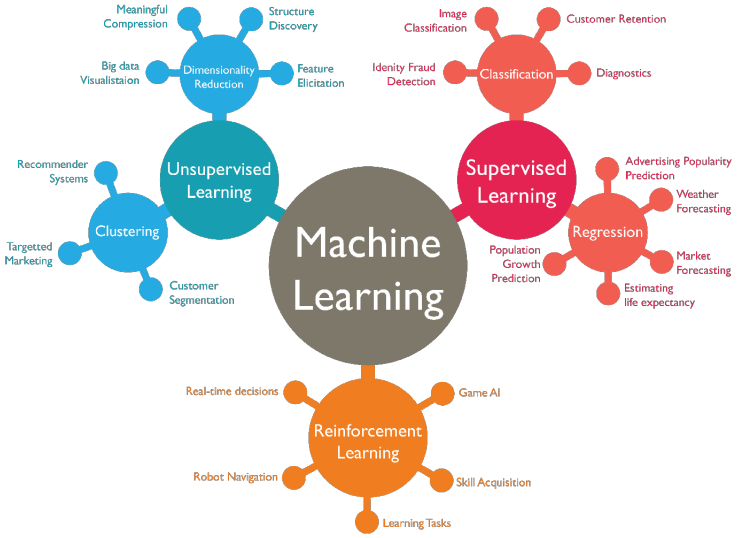

FIGURE 2.1 – Les types de machine learning

2.4 Les algorithmes du machine learning

2.4.1 Les réseaux de neurones artificiels

Les travaux sur les réseaux neuronaux artificiels, communément appelés "Neural Network", ont été motivés dès le début par la reconnaissance du fait que le cerveau humain calcule d'une manière totalement différente de celle de l'ordinateur numérique conventionnel. Le cerveau est un ordinateur (système de traitement de l'information) hautement complexe, non linéaire et parallèle.

Il a la capacité d'organiser ses constituants structurels, appelés neurones, de manière à d'effectuer certains calculs (par exemple, la reconnaissance des formes, la perception et le contrôle moteur) plusieurs fois plus rapidement que l'ordinateur numérique le plus rapide existant aujourd'hui. Considérant, Plus précisément le cerveau accomplit couramment des tâches de reconnaissance perceptive (par exemple, reconnaître un visage familier dans une scène inconnue) en 100-200 ms environ, alors que des tâches beaucoup moins complexes prennent beaucoup plus de temps sur un ordinateur puissant[14].

Le réseau neuronal artificiel est un système de traitement de données dont les neurones (éléments de processus) sont les bases des réseaux neuronaux artificiels. Les neurones ont 5 fonctions de base : entrées, poids, fonction de sommation, fonction d'activation et sortie.

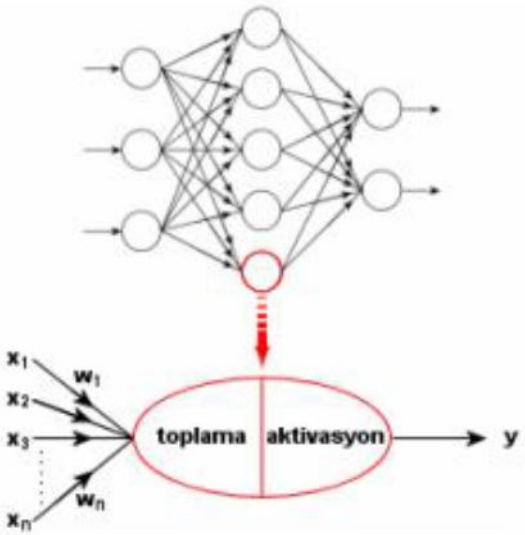

FIGURE 2.2 – La structure de réseau de neurone [12]

3. Entrées (x1, x2, ..., xn) : C'est la couche créée par l'utilisateur avec les échantillons de l'ensemble de données.
4. Poids (w1, w2, ..., wn) : Il indique quelle proportion des données d'entrée atteindra la sortie. Par exemple, le poids w1 indique dans quelle mesure l'entrée x1 affectera la sortie.

   Les valeurs des poids peuvent être modifiées, ce qui ne signifie pas que les entrées sont importantes ou non. Fonction de sommation : C'est la fonction qui est utilisée pour calculer l'entrée totale dans une cellule. Diverses fonctions sont utilisées pendant le calcul.

2.4.2 Arbres de décision

Un arbre de décision qui apprend à partir des données classées par induction est une structure de prise de décision.

Il s'agit d'un type d'algorithme d'apprentissage qui divise une grande quantité de données en petites portions en utilisant des étapes simples de prise de décision. à la fin de chaque division réussie, la similarité des éléments du groupe final augmente. Les arbres de décision, qui ont des caractéristiques descriptives et prédictives, sont l'un des algorithmes de classification les plus populaires car ils peuvent être facilement interprétés, intégrés aux bases de données et sont fiables.

Les arbres de décision ont trois structures : les noeuds de décision, les branches et les feuilles[12].

1. Noeuds racine : C'est un noeud qui n'a pas de branche antérieure et

qui peut créer une ou plusieurs branches.

Les noeuds racine montrent la variable dépendante et indiquent quelle variable sera utilisée pour la classification.

2. Noeud intérieur : C'est un noeud qui a une branche entrante et qui peut avoir deux branches sortantes ou plus.

3. Noeuds feuilles ou terminaux : Ce sont les noeuds qui ont une branche entrante mais pas de branche sortante.

   Il s'agit d'une structure qui montre le résultat du test entre les feuilles et les noeuds, et qui a pour rÃ´le de déterminer les groupes à définir. Si la classification n'est pas terminée à la fin de la branche, un noeud de décision émerge.

   La place des noeuds à l'extrémité de chaque branche est appelée profondeur. L'utilisateur peut déterminer le nombre de profondeurs en analysant l'adéquation de l'arbre de décision au jeu de données. Dans les arbres de décision, la profondeur et le nombre de groupes sont directement proportionnels.

2.4.3 Machines à vecteurs de support

Les machines à vecteurs de support (SVM) sont l'une des techniques de classification supervisée fondées par Cortes et Rapnik en 1995.

Les SVM sont un type d'algorithme machine qui fait des prédictions et des généralisations sur les nouvelles données en apprenant sur des ensembles de données dont la distribution n'est pas claire.

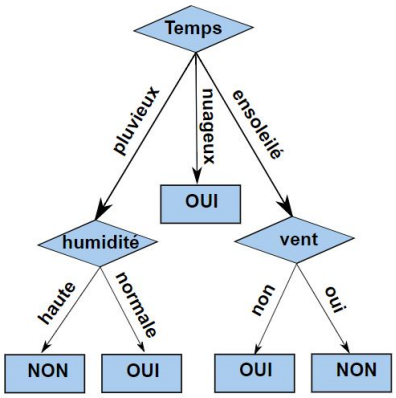

FIGURE 2.3 – Exemple d'un arbre de décision

Le principe principal des SVM est basé sur la recherche de l'hyperplan qui sépare le plus adéquatement les données de deux classes. Les machines à vecteurs de support sont divisées en deux catégories basées sur la classification que l'ensemble de données est séparé linéairement et non linéairement . Cas linéairement séparable : Avec les SVM, on vise à séparer les échantillons de deux classes qui sont généralement représentés avec les étiquettes (-1, +1) avec deux hyperplans différents les plus appropriés à l'aide de la fonction de décision générée à la fin des données d'apprentissage. Ce processus est atteint en trouvant l'hyperplan qui rend la longueur entre les points les plus proches du SVM maximale.

L'hyperplan qui rend la frontière maximale, l'hyperplan optimal et les points limitant la frontière sont appelés vecteurs de support [12].

2.4.4 Naïve Bayes

La classification Naïve Bayes est un type de classification qui est utilisé pour étiqueter les données en utilisant des méthodes statistiques.

Elle est préférée dans les problèmes de classification car elle est facile à appliquer.

En général, dans la classification de Bayes, il s'agit de calculer les valeurs de probabilité des effets de chaque critère sur le résultat. La méthode Naïve Bayes calcule la probabilité conditionnelle de la classe à laquelle appartiennent les données afin d'estimer la probabilité de la classe à laquelle appartiennent les données.

2.4.5 La régression logistique

La régression logistique est une méthode de classification qui modélise la relation entre plus d'une variable indépendante et une variable dépendante.

Il s'agit d'une méthode de régression avancée qui a gagné en popularité dans les sciences sociales aujourd'hui ; cependant, elle était davantage utilisée dans les sciences médicales par le passé.

2.5 Conclusion

Dans ce chapitre, nous avons vu la description et le principe de fonctionnement des différents types de l'apprentissage automatique, ainsi que les

définitions, les avantages et les inconvénients des algorithmes de l'apprentissage supervisé les plus utilisées dans la détection et la réparation des bugs de programmes. Dans le chapitre suivant, nous allons présenter la conception de notre solution pour le problème.

Chapitre 3

Conception du système

3.1 Introduction

Dans ce chapitre nous aborderons une description générale de notre programme, en mettant en évidence son coté conceptuel du pré-traitement qui constitue une étape fondamentale , ensuite nous détaillerons chaque phase en citant les principaux algorithmes et techniques utilisées.

3.2 Architecture générale

Notre programme se base sur l'utilisation du Machine Learning pour détecter les erreurs dans un programme écrit en JavaScript. Le système prend en entrée une base brute des identificateurs de JavaScript et les transforme en une base de caractéristiques utilisable par la phase d'apprentissage. Cette transformation est appelée pré-traitement, elle effectue une série d'opérations telles que le nettoyage, le filtrage et l'encodage. La base pré-traitée est subdivisée en deux partie ; une pour l'entraînement et l'autre pour le test. Le module d'entraînement utilise la base d'entraînement et un algorithme d'apprentissage pour fournir un modèle de décision qui est appliqué sur la base de test. Si le modèle est accepté, c-à-d a pu atteindre un taux de reconnaissance acceptable, il sera conservé et utilisé par le module d'utilisation et l'entraînement se termine. Dans le cas contraire, les paramètres de l'algorithme d'apprentissage sont révisé dans le but d'améliorer le taux de reconnaissance.

Nous avons supposé quand notre programme trouve une variable dans une instruction, il nous declare comme il ya une erreur.

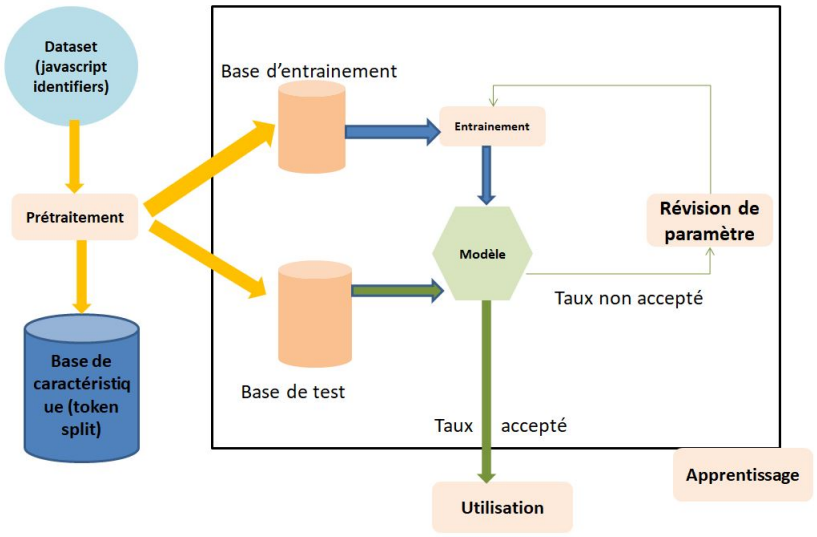

Figure 1.4. Architecture générale

3.3 Architecture détaillée

Notre objectif est d'extraire les meilleures caractéristiques permettant de détecter une variable dans une instruction. nous commençont par le prétraitement des données du dataset brute consiste àsélectionner et à préparer un ensemble de données d'entrainement.Ces données seront utilisées pour nourir le modèle de machine learning pour apprendre à résoudre le probléme pour lequel il est conçu. Dans notre cas les données de dataset sont subdivisées en trois catégories,aprés nous avons fait une étude sur notre dataset .

Apprentissage :il regroupe deux modules, l'entraînement et la validation

utilisant chacun une partie de la base des caractéristiques subdivisée en deux parties, base d'entraînement et base de test. Le module d'entraînement utilise la base d'entraînement pour fournir un modèle de décision tandis que le module de validation utilise la base de test pour mesurer la performance du modèle fourni. L'ensemble de test est nécessaire pour une évaluation impartiale du modèle final.

Dans les cas moins complexes, il est possible de travailler uniquement avec les ensembles d'entraînement et de test[15].

3.3.1 Entraînement :

L'ensemble d'entraînement est utilisé pour entraîner, ou ajuster,notre modèle. Par exemple, l'utilisation de l'ensemble d'apprentissage pour trouver les poids optimaux, ou coefficients, pour la régression linéaire, la régression logistique ou les réseaux neuronaux[15]. pour entraîner notre modèle, nous avons choisis la méthode que nous avons déjàprésenté dans le chapitre précédent ; les réseaux de neurones qui sont des algorithmes se présentant sous la forme d'un réseau à plusieurs couches,La premiére couche permet l'ingestion des données ,une ou plusieurs couches cachées tirent des conclusions à partir des données ingérées et la dernière couche assigne une probalité à chaque conclusion (l'algorithme de régression) .

3.3.2 Validation :

L'ensemble de validation est utilisé pour une évaluation non biaisée du modèle lors du réglage des hyperparamètres. Par exemple, lorsque le nombre optimal de neurones dans un réseau neuronal ou le meilleur noyau pour une machine à vecteurs de support, Pour chaque réglage envisagé des hyperparamètres, l'ajustement de modèle avec l'ensemble d'apprentissage et l'évaluation de ses performances avec l'ensemble de validation[15].

Le résultat de l'entraînement est un modèle ou pattern, qui représente l'analyse des données et leur transformation en informations utiles, en établissant des relations entre elles. En supposant que le modèle est une fonction mathématique y=f(x),ou x est la feauture,y est le label et f(x) le modèle.

Utilisation :C'est la dernière phase et la plus importante dans notre système. Apès être arrivé au meilleur taux de reconnaissance, ou après avoir construit le meilleur modèle dans la phase précédente, nous devons l'utiliser pour prédire l'occurance d'une variable dans une instruction.

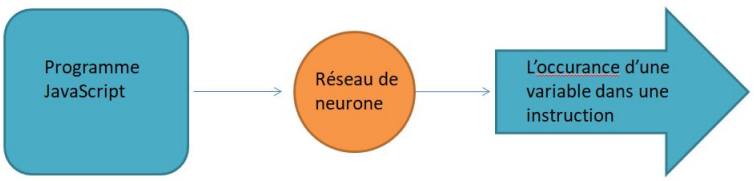

Figure 1.5. L'utilisation des réseaux de neurone dans notre système

3.4 Conclusion

Ce chapitre a décrit la conception de notre programme et il a présenté la démarche suivie dans ses différentes phases. Dans le chapitre suivant, nous allons décrire le fonctionnement de notre programme et les résultats obtenus.

Chapitre 4

Implémentation et résultats

4.1 Introduction

En informatique l'implémentation désigne mise en ouvre , ou la réalisation donc l'objectif de ce chapitre est de présenter les outils (les logiciels, les langages, les bibliothèques et les données utilisés dans notre programme, ensuite nous discuterons les résultats.

4.2 Outils utilisées

4.2.1 Logiciels utilisés

Dans notre travail nous avons utilisé l'environnement

Spyder(python),Pour l'entraînement et la validation est une environnement de développement scientifique Python, intégré (IDE) gratuit qui est inclus dans Anaconda. Il comprend des fonctions d'édition, de test interactif, de débogage et d'introspection.

Il est adapté á la programmation scientifique en python,ainsi qu'à la science des données et á l'apprentissage automatique.En fait,c'est l'un des meilleurs outils à cet effet.

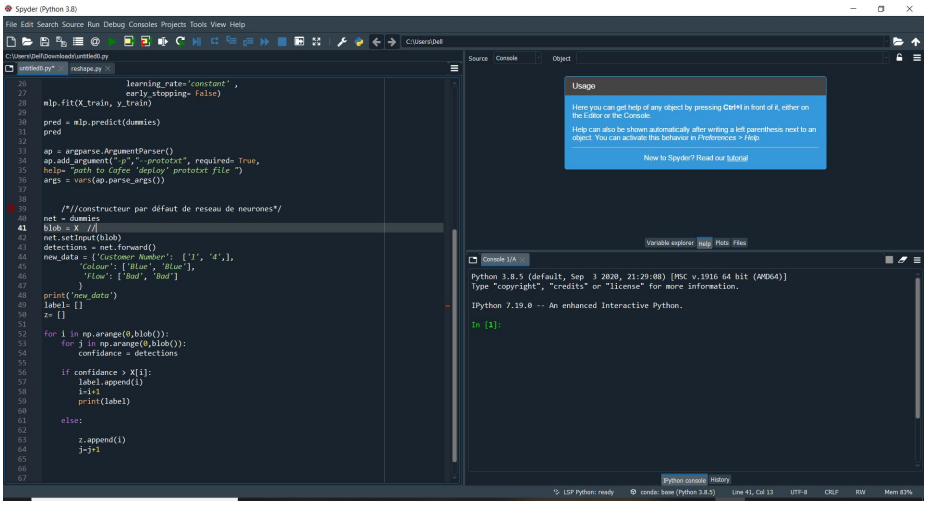

Figure 1.3.Spyder(Python)

4.2.2 Langages utilisés

Dans notre travail nous avons utilisé le langage Python en premier lieu pour le pré-traitement.

Python est un langage de programmation interprété,multi paradigme et multiplatforme. Il favorise la programmation impérative structuré, fonctionnelle et orienté objet.

Nous avons utilisé python parceque il offre un code concis et lisible. Alors que des algorithmes complexes et des flux de travail polyvalents sont à la base de l'apprentissage automatique et de l'IA,la simplicité de Python permet aux développeurs d'écrire des systèmes fiables. Le code Python est compréhensible par les humains,ce qui facilite la construction de modèles pour l'apprentissage automatique.

4.2.3 Bibliothèques utilisées

Dans chaque phase de notre travail nous avons eu besoin d'intégrer quelques bibliothèques,d'abord, en pré-traitement où nous avons utilisé deux bibliothèques .

1. Import train_test_split :
2. Multi-layer Perceptron (MLP_Regressor) :

   Le perceptron multicouche (MLP) est un algorithme d'apprentissage supervisé qui apprend une fonction $f(\cdot) : R^m \rightarrow R^o$ en s'entraînant sur un ensemble de donné es,$X = x_1, x_2, ..., x_m$, et une cible y,

   il peut apprendre un approximateur de fonction non linéaire pour la classification ou la ré gression. Il se distingue de la ré gression logistique par le fait qu'entre la couche d'entré e et la couche de sortie, il peut y avoir une ou plusieurs couches non liné aires, appelé es couches caché es. La figure4.3.2 montre un MLP à une couche caché e avec une sortie scalaire[16].

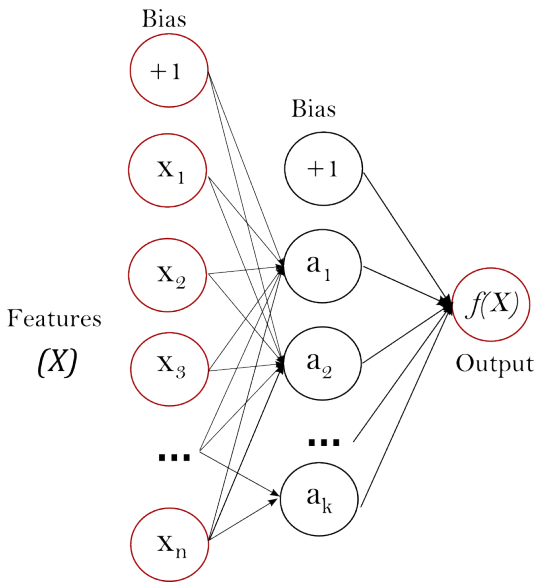

Figure 4.2.3Une couche caché MLP.

La couche la plus à gauche, appelé couche d'entrée, est constituée d'un ensemble de neurones $\{x_i|x_1, x_2, ..., x_m\}$représentant les caractéristiques d'entrée. Chaque neurone de la couche cachée transforme les valeurs de la couche précédente par une sommation linéaire pondérée $w_1x_1+w_2x_2+...+w_mx_m$ suivie d'une fonction d'activation non linéaire $g(\cdot) : R \rightarrow R$ comme la fonction tan hyperbolique. La couche de sortie reÃ§oit les valeurs de la dernière couche cachée et les transforme en valeurs de sortie.

Les avantages du Perceptron multicouche sont les suivants :

(a) Capacité à apprendre des modèles non linéaires.

(b) Capacité d'apprendre des modèles en temps réel (apprentissage en ligne) à l'aide de partial_fit.

Les inconvénients du Perceptron multicouche (MLP) sont les suivants :

(a) Les MLP à couches cachées ont une fonction de perte non convexe où il existe plus d'un minimum local. Par conséquent, différentes initialisations de poids aléatoires peuvent conduire à une précision de validation différente.

(b) MLP nécessite le réglage d'un certain nombre d'hyperparamètres tels que le nombre de neurones cachés, de couches et d'itérations.

MLP est sensible à l'échelle des caractéristiques.

3. MLPClassifier : La classe MLPClassifier met en oeuvre un algorithme de perceptron multicouche (MLP) qui s'entraîne par rétropropagation.

   MLP s'entraîne sur deux tableaux :

   le tableau X de taille (n_samples, n_features), qui contient les échantillons d'entraînement représentés sous forme de vecteurs de caractéristiques en virgule flottante ,

   Et le tableau y de taille (n_samples,), qui contient les valeurs cibles (étiquettes de classe) pour les échantillons d'entraînement :

4. Régression La classe MLPRegressor met en oeuvre un perceptron multicouche (MLP) qui s'entraîne en utilisant la rétropropagation sans fonction d'activation dans la couche de sortie, ce qui peut également ître considéré comme l'utilisation de la fonction d'identité comme fonction d'activation. Par conséquent, il utilise l'erreur carrée comme fonction de perte, et la sortie est un ensemble de valeurs continues.

   MLPRegressor prend également en charge la régression multi-sortie, dans laquelle un échantillon peut avoir plus d'une cible.

L'importance du fractionnement des données (Splitting)

L'apprentissage automatique supervisé consiste à créer des modèles qui mettent précisément en correspondance les entrées données (variables indépendantes ou prédicteurs) et les sorties données (variables dépendantes ou réponses).

La façon dont vous mesurez la précision de votre modèle dépend du type de problème que vous essayez de résoudre. Dans l'analyse de régression, vous utilisez généralement le coefficient de détermination, l'erreur quadratique moyenne, l'erreur absolue moyenne ou des quantités similaires. Pour les problèmes de classification, vous utilisez souvent l'exactitude, la précision, le rappel, le score F1 et des indicateurs similaires.

Les valeurs numériques acceptables qui mesurent la précision varient d'un domaine à l'autre. Vous devez évaluer le modèle avec des données fraîches qui n'ont pas été vues par le modèle auparavant. Vous pouvez y parvenir en divisant votre ensemble de données avant de l'utiliser[15].

Sous-assignation et sur-assignation : Le fractionnement d'un ensemble(Splitting) de données peut également être important pour détecter si votre modèle souffre de l'un des deux problèmes très courants que sont l'underfitting et le overfitting :

1. L'underfitting(Sous-assignation) :est généralement la conséquence de l'incapacité d'un modèle à encapsuler les relations entre les données. Par exemple, cela peut se produire lorsqu'on essaie de représenter

des relations non linéaires avec un modèle linéaire. Les modèles sous-adaptés auront probablement de mauvaises performances avec les ensembles de formation et de test[15].

2. Le overfitting (sur-assignation) : se produit généralement lorsqu'un modèle a une structure excessivement complexe et apprend à la fois les relations existantes entre les données et le bruit. De tels modèles ont souvent de mauvaises capacités de généralisation. Bien qu'ils fonctionnent bien avec les données d'apprentissage, ils donnent généralement de mauvaises performances avec les données non vues (de test)[15].

4.2.4 Datasets

Comme n'importe quel système utilisant le Machine Learning, nous avons besoin d'un dataset pour l'entrainer sur les données en utilisant ses algorithmes pour prédire la classe des données de la base de test. Le dataset de bugs brute que nous avons utilisé et préparé, chaque information a été transformée en un vecteur de caractéristiques et enregistrée dans un fichier CSV. Le fichier est ensuite subdivisé en deux parties, une consacrée à l'entraînement et l'autre au test et validation. Après la validation, nous somme passées à l'étape de l'utilisation qui consiste à prédire l'occurence de variable dans une instruction. à notre connaissance, et d'après nos recherches un dataset publique pour un type précis de bugs de programme contenant suffisamment de caractéristiques n'existe pas, ce qui nous a poussé à construire notre propre base.Cette dataset a été obtenue du site Kaggle qui est une plate-forme Web organisant des compétitions en science des données. Lien du Dataset :

$https : //www.kaggle.com/mbooth/javascript - identifiers?select = repos2ids_JavaScript.csv$

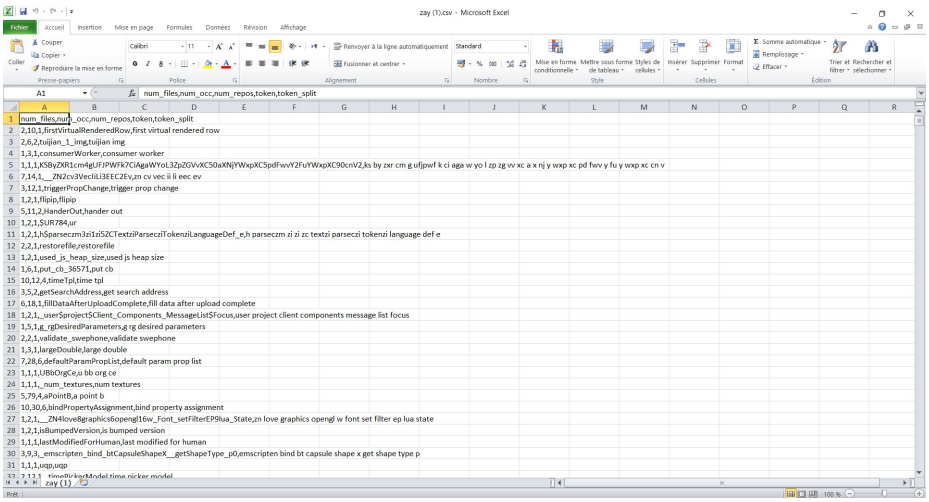

Figure 4.2.5. Une partie de notre dataset

4.2.5 Résultats Obtenus :

Le résultat de prétraîtement (Token split) on a gardé les colonnes de tokenSplit ,On a transformé les instructions de dataset en une matrice(dummies) (Figure 4.2.5.1)aprés on a pris un seul échantillon (token_split_af) sur lequel on a fait l'étude.

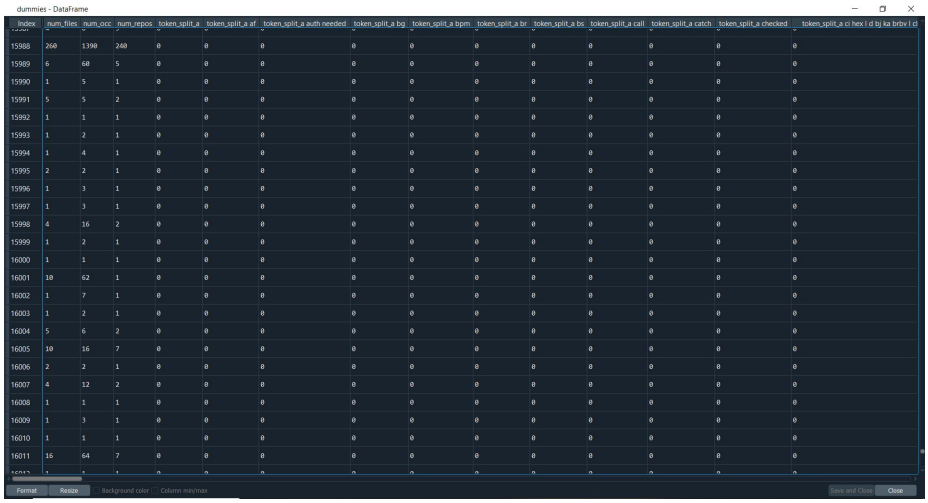

Figure 4.2.5. dummies(dataframe)matrice

La figure4.2.5 montre le résultat de l'exécution des statistique sous forme d'un plot ,la visualisation de plot Les points en blue montrent la relation entre les colonnes de la matrice(X)et l'occurence. le point isolé dans le plot nous montre que le programme à détecter un seul occurance d'une variable dans une instruction et d'aprè notre hyothèse ca peut empécher un bug.

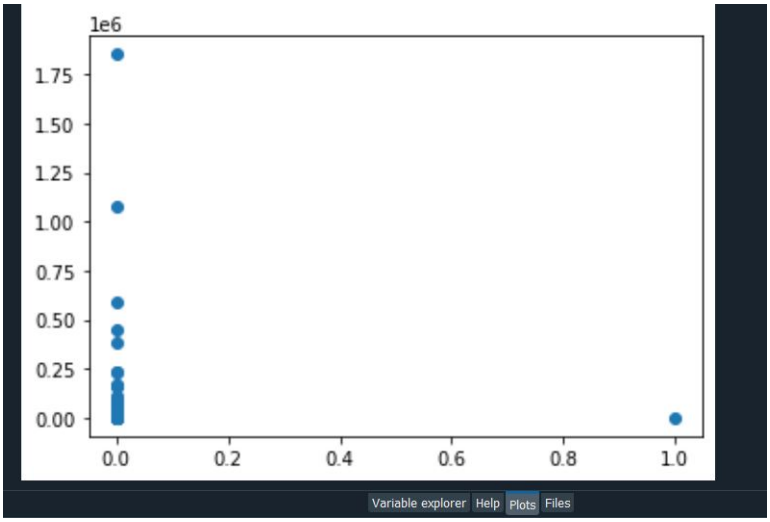

Figure 4.2.5. Plot de la visualisation de la détection des erreurs.

4.3 Conclusion

Nous avons détaillé dans ce chapitre les différents outils utilisés pour la mise en oeuvre de notre proposition ainsi que les résultats obtenus et leur discussion.

Conclusion générale

Le test logiciel est toujours un processus pénible, long, et coûteux. Des études récentes ont montré que les activités de débogage représentent souvent environ 50% des coût de développement. De nombreux facteurs contribuent au coût du débogage, mais le plus important est l'effort manuel considérable qui est encore nécessaire pour identifier et éliminer les défauts. Ce travail présente un outil de détection des erreurs dans les programmes en utilisant le machine learning. Nous avons exploiter le domaine fructueux du machine learning à la détection des erreurs des programmes. Vue l'indisponibilité de datasets adaptée à notre problématique, nous avons utilisé une dataset (JavaScript Identifiers) et nous avons fait l'hypothèse que l'occurrence d'une variable dans une instruction est une erreur. Nous avons utilisé pour l'implémentation de notre solution les réseaux de neaurones.

Perspectives :

Nous envisageons l'utilisation du machine learning dans la proposition de correction des bugs de programmes.

Annexe

Dans cette partie nous présontons le code source de notre programme qui sert à détecter la présence d'une variable dans une instruction comme un bogue logiciel.

La première partie de code source pésente les différentes bibliothèques utilisées dans notre programme :

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 import matplotlib.pyplot as plt
5 from sklearn.neural_network import MLPClassifier
6 from sklearn.neural_network import MLPRegressor
7 import argparse
8
9
```

Cette partie permet de lire le fichier (zay.csv) qui présente l'ensemble de données (la dataset des identificateurs du JavaScript).

La fonction "drop" élimine les instructions et nous gardons juste l'ensemble "Token_Split" (Les colonnes)

"Dummies" représente une matrice de l'ensemble des instructions de l'ensemble "Token_Split"

telque : "X" :un échantillon "af" de "Token_Split"

"y" :représente le numéro d'occurence d'une variable.

"Plt.scatter" Pour créer un plot qui visualise la relation entre les variables "X" et l'occurrence "y".

```
df = pd.read_csv(r"C:\Users\Dell\Desktop\zay.csv")

df.drop(['token'], axis='columns' ,inplace=True)

dummies = pd.get_dummies(df, columns=['token_split'])
X=dummies[['token_split_a af']]
y=dummies['num_occ']
plt.scatter(X,y)
```

Dans cette partie nous divisons notre dateset en deux :

1. Training sets .
2. Test sets .

La fonction "fit" Ajuste le modèle à la matrice de données . Après l'ajustement (apprentissage), le modèle peut prédire les étiquettes des nouveaux échantillons "dummies" avec la fonction "predict".

```
X_train , X_test , y_train , y_test = train_test_split(X,y,
    test_size = 0.2)
mlp = MLPRegressor(activation='tanh', solver='sgd' ,
learning_rate='constant' ,
early_stopping= False)
mlp.fit(X_train, y_train)

pred = mlp.predict(dummies)
pred
```

"Blob" présente tout les variables détectés, ils seront stockés dans la variable "detection"

```
ap = argparse.ArgumentParser()
ap.add_argument("-p","--prototxt", required= True,
help= "path to Cafee 'deploy' prototxt file ")
args = vars(ap.parse_args())

net = dummies
blob = X
net.setInput(blob)
detections = net.forward()
new_data = {'Customer Number':  ['1', '4',],
     'Colour': ['Blue', 'Blue'],
     'Flow': ['Bad', 'Bad']
}
print('new_data')
label= []
```

```
16 z= []
17
18 for i in np.arange(0,blob()):
19 for j in np.arange(0,blob()):
20 confidance = detections
21
22 if confidance > X[i]:
23 label.append(i)
24 i=i+1
25 print(label)
26
27 else:
28
29 z.append(i)
30 j=j+1
```

Bibliographie

[1] Seyyid Ahmed MEDJAHED, Analyse des méthodes d'apprentissage á base de noyaux . Application au diagnostic et á la classification des cellules cancéreuses.

[2] Machinelearningmastery, www.machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms, consulté le : 02/05/2021. [3] Lebigdata , https ://www.lebigdata.fr/machine-learning-et-big-data , consulté le : 05/05/2021

[4] Inria https ://www.inria.fr/fr/repairnator-un-robot-autonome-pour-reparer-les-bugs-informatiques,consulté le : 26/05/2021.

[5] L. Gazzola, D. Micucci and L. Mariani, "Automatic Software Repair : A Survey," in IEEE Transactions on Software Engineering, vol. 45, no. 1, pp. 34-67, 1 Jan. 2019, doi : 10.1109/TSE.2017.2755013.

[6] Xia, X., Lo, D., Shihab, E. et al. Automatic, high accuracy prediction of reopened bugs.

Autom Softw Eng 22, 75â109 (2015). https ://doi.org/10.1007/s10515-

014-0162-2

[7] Catolino, G., Palomba, F., Zaidman, A., Ferrucci, F. (2019). Not All Bugs Are the Same : Understanding, Characterizing, and Classifying the Root Cause of Bugs. ArXiv, abs/1907.11031.

[8] Rodriguez-Pérez, G., Robles, G., Serebrenik, A. et al. How bugs are born : a model to identify how bugs are introduced in software components. Empir Software Eng 25, 1294â1340 (2020).

https ://doi.org/10.1007/s10664-019-09781-y

[9] L. Gazzola, D. Micucci and L. Mariani, "Automatic Software Repair : A Survey," in IEEE Transactions on Software Engineering, vol. 45, no. 1, pp. 34-67, 1 Jan. 2019, doi : 10.1109/TSE.2017.2755013.

[10] Frei, R., McWilliam, R., Derrick, B. et al. Self-healing and self-repairing technologies. Int J Adv Manuf Technol 69, 1033â1061 (2013).

https ://doi.org/10.1007/s00170-013-5070-2

[11]Thomas Polacsek. Vérification, validation, certification : approches formelles et informelles pour établir la correction des artefacts et des logiciels. Systèmes embarqués. Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), 2019. fftel-02283804v2f

[12] Çelik, Özer. (2018). A Research on Machine Learning Methods and Its Applications. 10.31681/jetol.457046.

[13] Saker Amine, Christel Dartigues-Pallez, Rey Gaetan. CLASSIFICATION SUPERVISÉE DE DONNÉES PÉDAGOGIQUES POUR LA RÉUS-

SITE DANS L'ENSEIGNEMENT SUPÉRIEUR. [Rapport de recherche] I3S, Université Côte d'Azur. 2020. ffhal-02486729f

[14] Neural Networks and Learning Machines Third Edition ,Simon Haykin McMaster University Hamilton, Ontario, Canada

[15]RealPython, https ://realpython.com/train-test-split-python-data/the-importance-of-data-spl,Consulté le 24/06/2021.

[16]Scikit learn,

https ://scikit-learn.org/stable/modules/neural_networks_supervised.html,

consulté le 24/06/2021