Description détaillée du projet

Introduction

Le but du projet est de créer un système de recommandations qui permet de recommander les changements de code source. Dans ce projet, notre niveau de granularité serait le niveau classe.

Par exemple, si un développeur change une classe C_1 (C_1. java), le système de recommandation va lui suggérer les classes .java (eg. C_8, C_9, C_{45}) qui doivent changer aussi avec C_1.

Afin d'atteindre ce but, nous allons décomposer le projet en deux principales parties :

1. **Partie Mining (15% de la note finale du projet)**

Dans la partie Mining, vous allez choisir deux projets open-source JAVA de votre choix. Puis, vous allez extraire de ces projets les données relatives à l'historique du code source. Les fichiers de code source (. java) changés seront en fait l'input de l'algorithme de data mining que nous allons appliquer.

Exemples de projets JAVA open-source que vous pouvez utiliser : Eclipse, ElasticSearch, Rhino, ArgoUML, etc.

Les principales phases de cette première étape sont comme suit :

a. **Extraction des dépôts GitHub des projets étudiés (2%)**

Vous allez récupérer les informations importantes de l'historique du code source de vos projets y compris l'id du changement, le log concernant le changement, la date du changement, le type de modification effectué (ajout, modification, renommage ou suppression), le nom du fichier subissant une modification, etc.

Aide :

- Vous pouvez utiliser JGit pour extraire ces informations
- Pour une meilleure gestion des données, vous pouvez créer une base de données PostGreSQL pour stocker les informations

Puisque le projet met l'emphase sur la recommandation des changements de fichiers de code source .java, nous devons créer les transactions à base des fichiers .java changés pour chaque projet. Pour des raisons de simplification et tel qu'introduit dans les travaux antérieurs en Génie Logiciel (ex. Zimmermann et al., 2005), une transaction est un ensemble de fichiers changés dans un commit.

b. Filtrage des données (1%)

Certains filtrages peuvent s'appliquer à vos données pour éviter tout bruit qui peut impacter vos résultats. Par exemple :

- Un premier filtrage concerne les transactions n'étant liées qu'à un seul fichier.
- Un second filtrage que nous proposons va quant à lui éliminer les transactions possédant plus de cent fichiers (Zimmermann et al., 2005).

c. Création des données de training et de test (2%)

Dépendamment de l'historique de chaque projet étudié, vous allez choisir un historique de code source à examiner. Pour des raisons de simplification, nous vous proposons de ne pas dépasser 4 ans d'historique de code source.

Des transactions filtrées que vous avez construites en l'étape 2, vous allez sélectionner les mille dernières transactions de chaque projet comme vos données de test. Et le reste des transactions constituera vos données de training qui seront passées à votre algorithme de data mining pour générer les règles d'associations.

d. Application de l'Algorithme de Data Mining (5%)

Nous proposons l'algorithme le plus simple et largement utilisé dans l'état de l'art en génie logiciel, i.e. Apriori par (Agrawal & Srikant, 1994).

Aide : vous pouvez utiliser l'environnement R pour appliquer Apriori. Pour ce faire, il faut installer le package 'arules'. Les commandes essentielles sont disponible sous :

https://cran.r-project.org/web/packages/arules/arules.pdf

e. Rédaction de la partie Mining (5%)

Vous allez résumer et soumettre via Moodle la partie Mining en un document (Cf. template projet.doc) qui ne dépasse pas 4 pages y compris les références.

2. Partie Empirique (15% de la note finale du projet)

a. Conception de l'étude empirique (3%)

Suivant le cadre de Basili vu en cours.

b. Application des méthodes analytiques (5%)

Calcul de la précision, rappel, et F-mesure.

Aide :

Consulter l'explication de ces notions sur le lien suivant : https://en.wikipedia.org/wiki/Precision_and_recall

c. Résultats et discussion (2%)

Génération des résultats pour chaque projet et interprétation.

d. Rédaction de la partie Empirique (5%)

Remarque :

Ce travail a été déjà fait et publié en la littérature du Génie Logiciel. On propose, dans le contexte de ce cours, sa réplication de façon largement simplifiée aux étudiants pour qu'ils appliquent de façon pratique les notions vues en classe et pour tirer bénéfice de cette expérience dans leur travail en industrie ou en recherche.

Pour plus de détails, vous pouvez consulter l'article suivant :

Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, Andreas Zeller: Mining Version Histories to Guide Software Changes. ICSE 2004: 563-572