Laboratoire MGL869 :

Utilisation de l'apprentissage machine pour prédiction des bogues

Automne 2024

1. **Objectif du laboratoire :**

Autant qu'ingénieur DevOps dans le projet Hive, vous cherchez à introduire la qualité à la source en faisant une prédiction des bogues avant de publier votre nouvelle version de code. Connaître les fichiers qui ont plus de chance d'avoir des bogues vous permettra de prioriser les fichiers à tester surtout durant les derniers jours avant de réaliser une nouvelle version de code. En particulier, vous cherchez à développer un modèle d'apprentissage machine qui vous donne la liste des fichiers qui sont les plus susceptibles d'avoir un bogue et sur lesquels vous aller prioriser vos tests avant de réaliser une nouvelle version du logiciel Hive.

Pour ceci, vous allez utiliser les sources de données suivantes de Hive :

- Le repértoire Jira : https://issues.apache.org/jira/projects/HIVE/issues/HIVE-13282?filter=allopenissues
- Le repértoire de code : https://github.com/apache/hive
- L'outil pour collecter des métriques du code : https://www.scitools.com/student

2. **Collection de données (25 points) :**

Le but de cette première étape est de collecter les données. En particulier, l'identification de quel fichier contient un bogue dans quelle version de Hive et les variables à utiliser dans le modèle. Pour une question de simplification, il est demandé d'utiliser **juste les données des versions à partir de la version 2.0.0** de Hive.

   a. **Collection des fichiers qui contiennent un bogue :**

L'objectif de cette étape est de savoir quel fichier de code (Java ou C++) contient un bogue et dans quelle version du logiciel Hive. Pour ceci, il faut utiliser le répertoire Jira qui contient des informations sur ce qui a été développé incluant la correction des bogues. Pour simplifier, cet exercice va se concentrer sur les bogues qui ont été déjà résolus et que leurs rapports contiennent les versions impactées. Personnaliser le filtre de la recherche de Jira pour ne sélectionner que les bogues qui impactent des versions déjà publiées.

À partir de Jira, vous pouvez obtenir quel bogue (identifié par un ID) impacte quelle(s) version(s) de Hive. En utilisant le répertoire de code, il faut identifier quels fichiers de code ont été modifiés pour résoudre un bogue. En particulier, les développeurs de Hive écrivent des messages de bonne qualité en sorte qu'ils mentionnent l'id du bogue qu'un commit adresse. Du coup, il vous a demandé que vous utilisez les traces de Git (git log) pour savoir quels fichiers ont été modifiés pour adresser un bogue.

Le résultat de cette étape est l'association entre les bogues et les fichiers de code qui contenaient ces bogues. En d'autres mots, le résultat est un fichier csv dont chaque ligne est un bogue (son ID sur la plateforme Jira), un fichier qui contenait le bogue.

b. Collection des variables indépendantes :

Pour prédire les bogues, il faut avoir des variables indépendantes qui caractérisent chaque fichiers de code juste avant de réaliser une nouvelle version de Hive et ceci est pour toutes les versions de Hive. Par exemple, le fichier A a une complexité X avant de réaliser la version 0.0.1 de Hive. Le même fichier aura une complexité Y avant de réaliser la version 0.2.2 de Hive. Même chose pour tous les autres fichiers de code Java ou C++ de Hive. Vous pouvez développer un algorithme pour identifier le dernier commit de chaque version de Hive en utilisant la date de réalisation des versions de Hive. Il faut prendre la version du code à un tel commit (git checkout). Il faut ensuite collecter les variables à utiliser dans le modèle pour chaque fichier au niveau du commit. D'une manière similaire à la Section 1.a, chaque fichiers aura différentes valeurs de variables d'une version de Hive à une autre. En particulier, il faut collecter pour chaque fichier/version les variables dans le tableau suivant :

Granularity	Metrics	Count
File	AvgCyclomatic, AvgCyclomaticModified, AvgCyclomaticStrict, AvgEssential, AvgLine, AvgLineBlank, AvgLineCode, AvgLineComment, CountDeclClass, CountDeclClassMethod, CountDeclClassVariable, CountDeclFunction, CountDeclInstanceMethod, CountDeclInstanceVariable, CountDeclMethod, CountDeclMethodDefault, CountDeclMethodPrivate, CountDeclMethodProtected, CountDeclMethodPublic, CountLine, CountLineBlank, CountLineCode, CountLineCodeDecl, CountLineCodeExe, CountLineComment, CountSemicolon, CountStmt, CountStmtDecl, CountStmtExe, MaxCyclomatic, MaxCyclomaticModified, MaxCyclomaticStrict, RatioCommentToCode, SumCyclomatic, SumCyclomaticModified, SumCyclomaticStrict, SumEssential	37
Class	CountClassBase, CountClassCoupled, CountClassDerived, MaxInheritanceTree, PercentLackOfCohesion	5
Method	CountInput_{Min, Mean, Max}, CountOutput_{Min, Mean, Max}, CountPath_{Min, Mean, Max}, MaxNesting_{Min, Mean, Max}	12

***Table 1**. Variables à collecter inspiré de "Yatish et al. Mining Software Defects: Should We Consider Affected Releases? ICSE 2019". Pour la section méthode, il suffira de collecter juste les 5 métriques de bases de Understand: CountInput, CountOutput, CountPath, MaxNesting. En d'autre mots, il faut collecter 46 variables.*

Pour collecter ces variables, il faut exécuter l'outil Understand (https://www.scitools.com/student). À noter que vous pouvez avoir une version gratuite de l'outil Understand autant qu'étudiants. On vous refère à la documentation de Understand pour savoir comment l'utiliser en ligne de commande : https://support.scitools.com/support/solutions/articles/70000582798-using-understand-from-the-command-line-with-und

Votre fichier csv doit ressembler à

Version,CommitId,Fichier,AvgCyclomatic,AvgCyclomaticModified,...
2.1.0,id1,f1,21,41,209,...
2.1.0,id1,f2,91,4,9,...
2.5.0,id2,f1,31,51,19,...
2.3.0,id3,f2,41,41,9,...
…

c. Labellisation des données :

Ajoutez une colonne qui indique si un fichier A dans une version de Hive B contient un bogue ou pas, en utilisant les données de la Section 1.a.

3. Construction du modèle de prédiction (25 points) :

a. Implémentation du pipeline :

Tel que vu dans le cours, implémentez le pipeline pour entrainer et tester un modèle d'apprentissage machine pour prédire quel fichier aura un bogue avant de réaliser une nouvelle version de code. Pour ce laboratoire, on vous invite à utiliser et comparer "logistic regression" et "random forest". On cherche à comparer si les deux modèles auront des résultats différents en termes de performances et interprétabilité. À noter qu'il faut retirer les variables qui correspondent aux noms de fichiers, noms de versions, numéro de commits de vos données pour construire votre modèle d'apprentissage machine.

b. Évaluation de la performances du modèle :

Pour chacun des deux modèles, évaluer leurs performances. En particulier, calculer la valeur du AUC, de la précision et du rappel (recall). Vous pouvez utiliser un graphe de votre choix pour visualiser les résultats. Quel modèle est le plus performant ?

c. Interprétation du modèle :

Identifiez les variables les plus importantes pour la prédiction des fichiers qui contiennent un bogue. Visualiser l'impact de chaque variable en utilisant un nomogramme.

4. Analyser les résultats obtenus (20 points) :

En utilisant la liste des variables les plus importantes et le nomogramme, analysez les résultats obtenus en suggérant cinq actions à prendre pour éviter des bogues dans le système. Vous pouvez vous référer à des exemples concrets du projet de Hive. Justifiez vos réponses et vos choix.

5. Identifier les limitations du modèle (20 points):

Les pratiques et les difficultés changent avec le temps. En effet, les développeurs changent et améliorent leurs pratiques. De même, un logiciel devient de plus en plus volumineux et souvent compliqué à maintenir. Du coup, le contexte du projet en premières versions changent en comparant avec le même projet dans les dernières versions de Hive. Cependant, le même modèle ne peut pas garder une performance stable tout au long du cycle de vie du logiciel Hive.

Pour valider cette hypothèse, on vous demande de faire un modèle pour chaque version de Hive.

Pour cet exercice, ne considérez que les versions mineures et pas les patches. En fait, chaque numéro de version de Hive est composé de 3 chiffres qui respectent la structure suivante : <version majeure>.<version mineure>.<patch>. Par exemple, 2.4.1 correspond à la version majeure 2, version mineur 4 et le patch 1. Dans cet exercice, considérez juste les versions mineurs. En particulier, faites un modèle pour la version 2.0.0, un modèle pour la version 2.1.0 et ignorez tous les patches entre 2.0.0 et 2.1.0. De même jusqu'à la dernière version de Hive. Attention, ne pas utiliser des données qui datent d'avant 2.1.0 pour entraîner un modèle basé sur la version 2.1.0.

a. **Performances :**

Utiliser un graphe pour montrer comment les trois métriques de performances des modèles évoluent d'une version à une autre. Comparez et interprétez les résultats.

b. **Interprétabilité :**

Pour raison de simplification, comparer les variables les plus importantes juste des modèles de versions majeurs (modèle de version 2.0.0 vs modèle de 3.0.0). Intérpértez et analysez les résultats. Vous pouvez visualiser la distribution des variables qui changent. Est ce que Hive doit toujours utiliser le même modèle, est ce qu'il doivent mettre à jour leurs modèles ?

6. **Remises et dates limites (10 points)**:

La qualité de la présentation (5 points).
Qualité du rapport (5 points).

Date limites :

- **07 Novembre** - présentation du laboratoire.
- **05 Décembre** - remise du rapport finale.

Instructions sur le rapport finale :

Le rapport final doit contenir une section d'introduction, une section détaillant votre méthodologie, une question des résultats, une question de discussion des résultats et finalement une conclusion. **La qualité du rapport et de la rédaction compte pour 5 points. À ne pas dépasser 5 pages dans le rapport final.**

Évaluation :

Collection des données	**25**
Construction du modèle	**25**
Analyse des résultats	**20**
Limitation du modèle	**20**
Qualité de la présentation et du rapport	**10**
Total	**100%**