Université Paris 8

Master 2 Big Data et fouille de données

Logiciels libres et protection de données

Date : 02/01/2017

Résumé d'article «Can Developer-Module Networks Predict Failures in Open-Source GitHub Projects ?»

Réalisé par

Nabil Redha BELKHOUS

Numéro d'étudiant : 16705491

Enseignant

Ivaylo Ganchev

Table des matières

1 Introduction

Pour mon travail de lecture d'article, j'ai choisi d'étudier l'acticle « **Can Developer-Module Networks Predict Failures in Open-Source GitHub Projects ?** » ecrit par Bryan J. Muscedere et **Rafi Turas**[1].

Cet article met en avant un type de réseau social qui a été utilisé lors du développement de **Window Vista** et qui a donné des prédictions d'echec avec précision lors du développement de ce projet.

Il s'agit du **developer-module network** qui n'est que les interactions entre les developpeurs et les fichiers (fichiers binaires, historiques des commits, bugs notifiés, ...)

Cependant il a été utilisé uniquement sur le projet de Windows Vista. C'est pour cela que cette étude a voulu tester l'efficassité de ce type de réseau social avec 20 projets open source disponnible sur GitHub.

2 Projets sur les quel les métriques ont été relevées

Sur le projet **Windows Vista**, on a utilisé plusieurs métriques notemment :
— métriques de centralitées (Centrality metrics)
— métriques organisationnelles (organizational metrics)
Pour le projet **ECLIPSE** on a utilisé les **métriques organisationnelles**

Centrality metrics : (identifier les sommets les plus significatifs du réseau) des fichiers binaires pour quantifier la contribution des développeurs à ces fichiers afin de prédire les fichiers qui sont sujets aux bugs (bug-prone files)

organizational metrics : étudier la structure de l'equipe et la structure du projet. Généralement plus les structures sont similaires, plus le projet a de chance de réussire.

J'ai remarqué que le **developer-module networks** était constuit à partir de metriques de codes sources car sur les noeuds qui caractèrisent les développeurs notés D, des informations comme le nombre de commits ou le nombre de bugs fixés sont enregistrés.

3 Les hypothèses

1. Les noeuds du réseau (les fichiers) avec une haute centralité intermédiaire sont plus sujets aux bugs (bug-prone files) du fait de leur importance dans le projet.
2. Les noeuds du réseau (les fichiers) avec une haute centralité de proximité sont plus susceptible d'être sujets aux bugs (bug-prone files) car ils sont plus près des développeurs.
3. Les noeuds du réseau (les fichiers) avec un haut degré de centralité sont plus sujets aux bugs (bug-prone files) car il y a un grand nombre de développeurs qui modifient ces fichiers durant le développement du projet.

4 Les métriques qui ont été privilégiés

Les metrics **network centrality** font référence aux métriques qui décrivent les acteurs (développeurs et fichiers) les plus importants du réseau (Plus grand nombre de commits, plus grand nombre de bugs, les fichiers les plus utilisés, nombre d'utilisateurs d'un fichier, ...).

Dans cette famille de métrique, on a utilisé :

1. **Betweenness centrality ou centralité intermédiaire** : Elle est égale au nombre de fois qu'un sommet est sur le chemin le plus court entre deux autres nœuds quelconques du graphe. Un nœud possède une grande centralité intermédiaire s'il a une grande influence sur les transferts de données dans le réseau.

2. **Closeness centrality ou centralité de proximité** : Dans un réseau, la distance entre paires de nœuds est définie par la somme de la longueur de leur plus courts chemins. La centralité de proximité est l'inverse de cette somme. Ainsi, un nœud plus central a une distance plus faible à tous les autres nœuds.

3. **Degree centrality ou centralité de degré** : Elle est définie comme le nombre de liens incidents à un nœud (c'est-à-dire le nombre de voisins que possède un nœud).

5 Les outils statistiques qui ont été utilisés

Afin de valider les différents les hypothèses, plusieurs méthodes statistiques ont été utilisées.

— **Spearman Correlation** : qui a permit de valider que les centralités intermédiaires, de proximité et de degré étaient coorélés avec le nombre de commits et les commits qui ont fixés les bugs pour chaque projet.

— **Logistical Regression** : le but était de développer des modèles logiques à partir des métriques de centralitées qui vont être utilisées pour prédire si les fichiers dans un projet sont suceptiblesd'être sujet aux bugs (bug-prone files) ou non (non-bug-prone file).

6 La méthode pour la récolte et l'analyse des données

Pour cette étude, les auteurs ont choisi de travailler avec des projets open source disponible sur **GitHub**.

Ils ont mis en place un **developer-module networks** pour chaque projet pour calculer les **métriques de centralitées** et voir si ces metriques étaient corrélées avec le nombre de commits et le nombre de bugs fixés et surtout si les **métriques de centralitées** étaient capables de prédire les fichiers les plus sujets aux bugs (bug-prone files).

Dans cette étude l'accès aux données ne se fait pas directement sur les repositories des projets mais en utilisant le projet **BOA** qui a un **DSL (domain specific language)** qui permet facilement d'accéder aux **milliers** de **GitHub repositories** qui sont dans sa base de données à travers un langage de requetage.

En utilisant **BOA** les auteurs ont développé une requete appelée : **project search** qui permet de chercher des projets selon le nombre de **contributeurs**, **fichiers**, le ratio des **commits** et les **commits qui ont fixé les bugs**.

Ils ont aussi développé une fonction nommée : **contribution data** qui permet de récupérer les données nécessairees (une liste de fichiers, une liste de contributeurs et une liste de fichiers changés par chacun commit).

Ces deux fonctions aident à récupérer facilement les données pour constuire le **developer-module networks**

Une application a été dévlopper en Java et en Scala. **NetworkMine tool** contient deux modules : un résponsable de récuperer les données et construire le **developer-module networks** et un autre module qui va calculer toutes les métriques et lancer le calcul de la **corélation de Spearman** et la **régression logistique** pour valider les différents les hypothèses.

7 Résultats

Les auteurs ont décidé de tester leur outil **NetworkMine** avec quinze projets tirés de la base de données BOA. Les projets ont une **centaine de contributeurs**, **un millier de fichiers** et viennent de différents environements de développment.

Ce tableau montre quelques statistiques concernant les utilisateurs, les fichiers, les commits, et les bugs fixés

	Users	**Files**	**Commits**	**Bug Fixes**
Average	190.20	3268.47	27.30	4.69
Max	308	6683	4486	1472
Min	102	1448	1	0

Dans les deux tableaux suivants on peut voir les résultats de précisions et de rappels sur les trois projets qui ont donné les meilleurs résultats et les trois projets qui ont donné les plus mauvais résultats

Top Projects		**Precision**	**Recall**
Gerrit	Mean	0.797	0.951
	Median	0.797	0.951
Caf-Platform	Mean	0.753	0.816
	Median	0.753	0.813
System Core (Android)	Mean	0.715	0.815
	Median	0.740	0.818

Bottom Projects		**Precision**	**Recall**
Gem5	Mean	0.568	0.861
	Median	0.546	0.880
PhoneGap-Plugins	Mean	0.911	0.537
	Median	0.911	0.537
Google Closure Comp.	Mean	0.999	0.647
	Median	1.000	0.648

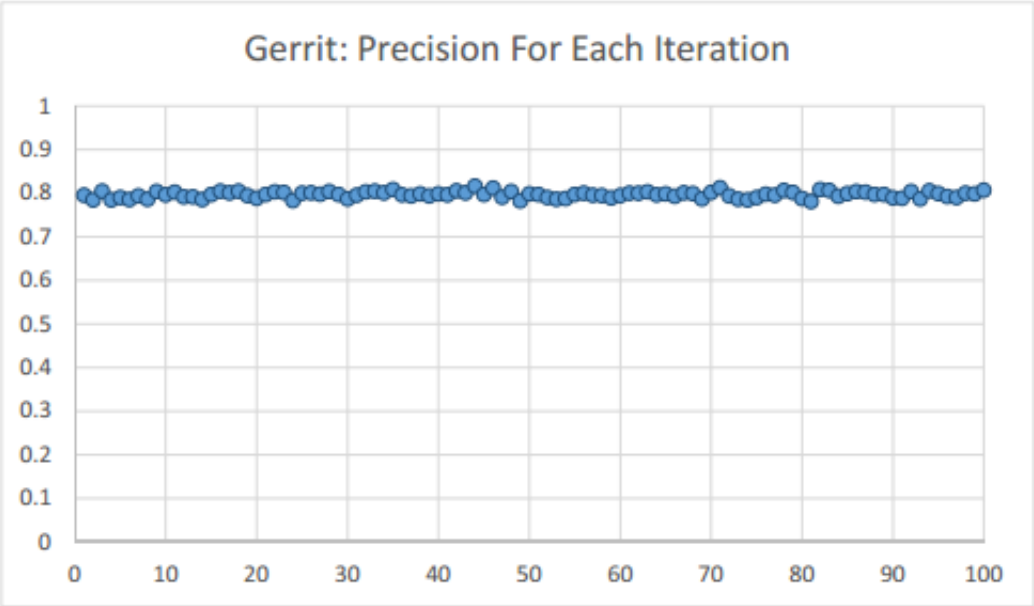
Le tableau suivant montre la correlation spearman moyenne pour tous les projets. Les valeurs supérieures à 0.5 denotent **une correlation positive** et les valeurs superieurs à 0.7 denotent **une correlation positive forte**

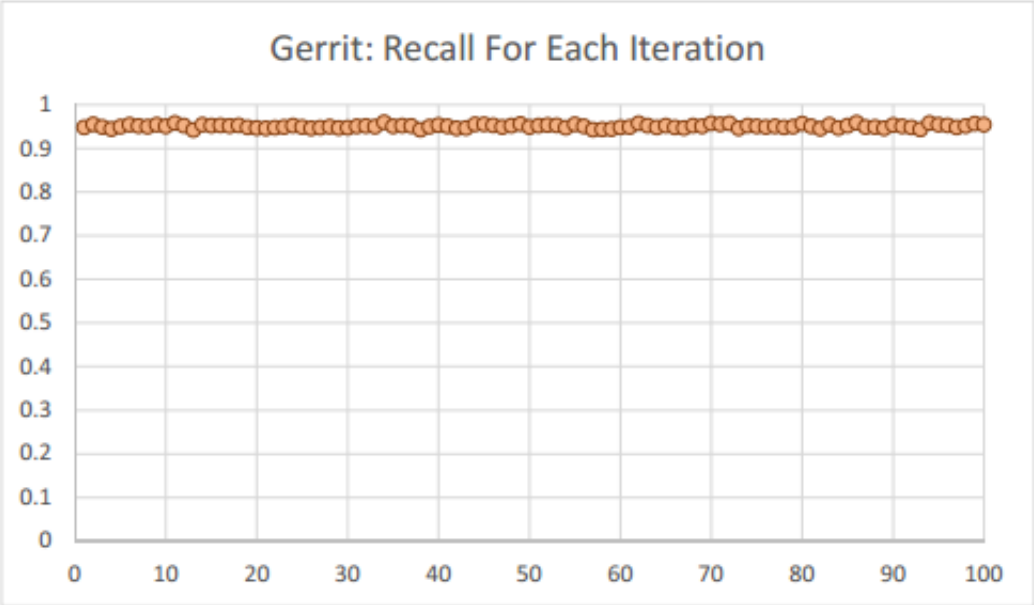
	Commits	**Bug Fixes**	**Betweenness**	**Closeness**	**Degree**
Commits	1.00	0.811	0.507	0.173	0.852
Bug Fixes		1.00	0.453	0.213	0.750
Betweenness			1.00	0.627	0.723
Closeness				1.00	0.459
Degree					1.00

Le tableau suivant montre la correlation spearman moyenne pour le projet **Caf-Platform** (en noir) et le projet **Dropwizard** (en rouge).

	Commits	Bug Fixes	Betweenness	Closeness	Degree
Commits	1.00 / 1.00	0.927 / 0.680	0.802 / 0.251	0.870 / -0.281	0.971 / 0.737
Bug Fixes		1.00 / 1.00	0.782 / 0.093	0.821 / -0.133	0.933 / 0.529
Betweenness			1.00 / 1.00	0.898 / 0.616	0.867 / 0.675
Closeness				1.00 / 1.00	0.926 / 0.332
Degree					1.00 / 1.00

Les figures suivantes montrent le résultats de la régression logistique pour le projet **Gerrit**. Le résultat de prédiction confirme ce qu'on avait obtenu dans le tableau numéro 2 avec les tops projects.





Références

[1] Muscedere Bryan J and Turas Rafi. Can Developer-Module Networks Predict Failures in Open-Source GitHub Projects ?, 2011