A search-based software engineering for defect prediction in ubuntu ecosystem

A search-based software engineering for defect prediction in ubuntu ecosystem

I Made Murwantara*, Sutrisno and Joseph

Informatics Dept, Faculty of Computer Science, Universitas Pelita Harapan

*made.murwantara@uph.edu

Abstract. A software defect may cost a lot for certain critical deployment such as communication and inter-process management that affects the CPU processes. A lot of time and efforts can be saved if we can predict the potential defect. In this study, we propose an approach to identify and predict the software defects that make use of Search-Based Software Engineering (SBSE) via a network model that collects its dataset from Ubuntu package repositories. We create a network model of package dependency that generated from the repository via a graph utility, namely debtree. In order to explicate the conflict between packages, we analyze the package inter-dependencies as related graph clusters. Since the massive number of dependencies, we use several Machine Learning methods to analyze and predict the potential software defect. Our investigation result shows that the SBSE technique has efficiently improved the defect identification process.

1. Introduction

Open source repository is a way of modern software distribution that are managed in packages. A Software in a package can be installed or removed in a random way by referring to their dependencies. In the Ubuntu package repository, package manager control most of all actions that related to installation, removal and administrative tasks, automatically, without user intervention. With this manner, a package configuration should be able to setup a matching package dependency in the direction to build and deploy an application.

A software repository, such as Ubuntu repository, consists of thousands of packages that evolves through out the years. Most of these packages have dependencies to create functionality as application that is built from the ecosystem. However, a package may have conflict to other packages in the repository because of certain reason. One of the reason is the update of functionality to enhance security and performance that may affect existing package dependencies.

In last decade Search-Based Software Engineering (SBSE) technique has been utilized to identify defect [1,2,3] and analyze software repository [4] in most open source ecosystem. Since Open source Operating System distribution repository, such Ubuntu and FreeBSD, may consists a large number amount of software packages or components, identifying defect in a repository become an interesting issue and has been explored by many research groups. Artho et al. [5] have investigated the inter-package conflict of the Debian repository. They found that with more detailed of package meta-data will prevent one third of conflict packages in a repository. Tan et al. [6] studied on software bug characteristic of open-source software in Linux kernal, Mozilla and Apache. Sharma et al. [7] developed a prediction model to find bugs within open-source repositories due to source code changes.

Boisselle et al [8] show that cross-distribution within Debian and Ubuntu has created duplicate bugs and spent a vast amount of lost time due to the delay for the community to fix the problem. In this work, we analyze and predict the defect within Ubuntu repository in a network model of package dependency. This paper aims to empirically investigate the relationship between bugs in the Ubuntu package repositories and their conflict dependency as stated in the packages control manager.

2. Method and materials

2.1. Creates Data Set

In this work, we create dataset from several versions of Ubuntu repositories, namely Ubuntu distribution version 12, 14 and 16. The steps to obtain dataset as follow, first, in a repository; a list of packages is created by retrieving all available package names that installed in a standard installation of a particular Ubuntu release. Then, we retrieve all the packages description from the package list. We make use of "debtree", the Ubuntu utility to retrieve and transform the package with its dependency into a network like model which is in the form of Graphviz, the network model shown in Figure 1. After that, we transform the network model into data set in csv format before we analyze it using several Machine Learning model. In our dataset, each package has its dependency is categorized into depends-on as 1, conflicts as 2 and suggest as 3. We interested with conflict packages, so that the dataset is transform only to have conflict information. The Decision Tree make split based on the complex parameter that is accepted by the R Rpart package.

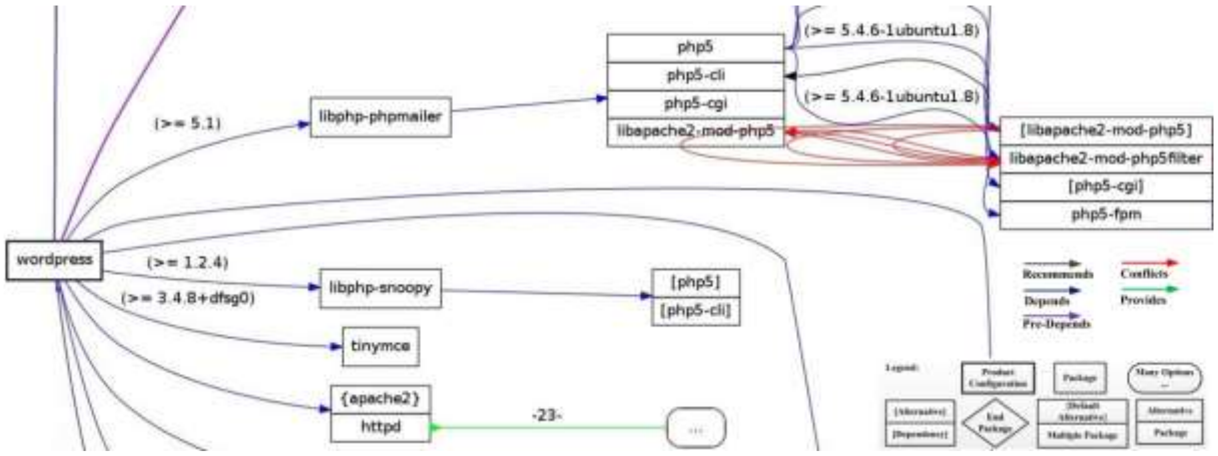

Figure 1. An Excerpt of Network Model Retrieve from Ubuntu Package Repository.

2.2. Method

We make use Classification Tree (CT) to analyse conflict between packages based on their relation of dependencies. The Regression Tree (RT) differs to CT in term of prediction value comparison on each node, as shown in Figure 2. In each node RT has two values as prediction output and percentage number of observation data. In order to optimize a tree and to reduce error we prune the Classification Tree (CT) and Regression Tree (RT). Pruning is done based on minimum split and parameter complexity of each model. Then, the result tree will have its parameter complexity that should minimize error. We choose the Complexity Parameter (CP), based on minimum xerror where it is the variable that consists of error prediction from a cross-validation for each split. We pick value 0.006894 as the CP to prune our Classification Tree (CT).

Pruning in CT presents a bigger tree with more accurate prediction. This is because of decision tree that make the reduction of split if it has bigger cross-validation error then the previous training. Pruning in Regression Tree (RT) is similar to the Classification Tree (CT). Based on the conditional probability of RT, we choose complexity parameter of 0.000910531 that has cross-validation error minimum of 0.819192.

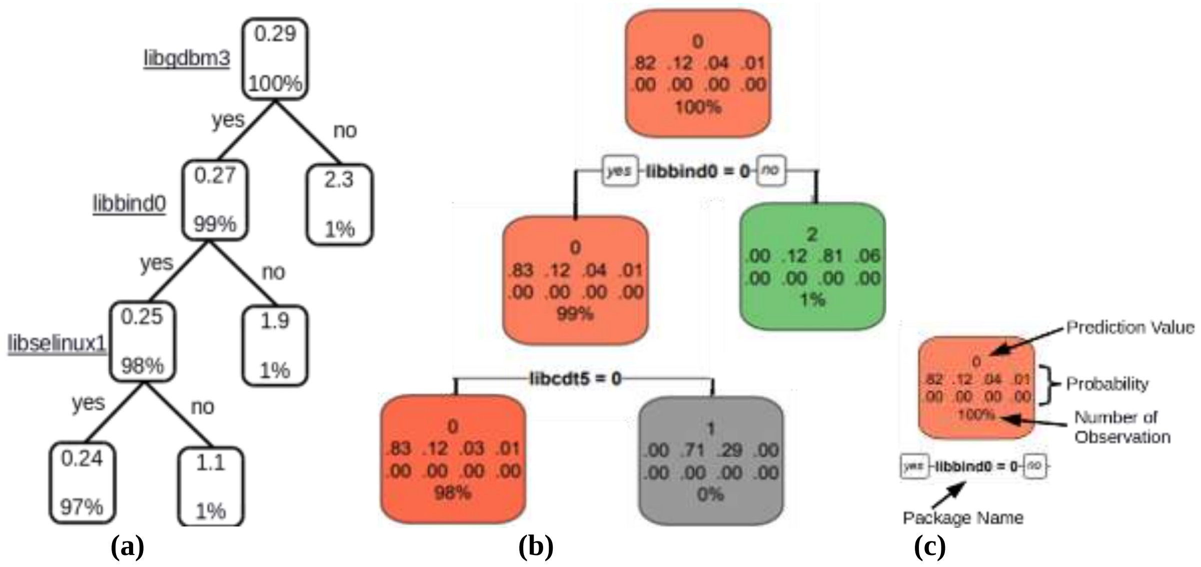

Figure 2. Decision Tree of Linux Library, (a) Regression Tree; (b) Classification Tree with explanation in (c).

Random Forest (RF) is an ensemble tasks that combine classification and regression that construct decision trees from its nearest neighbor predictor. In our work, RF split point indicates the relation of package that contain node terminal or output prediction. However, RF cannot predict beyond the training data when uses regression. In order to decide the split of decision trees, rather than using all input variable, a randomly chosen input is selected. This technique can help to improve the prediction. Therefore, it is worth investigating the RF result in our data set to have better prediction.

3. Results and discussion

3.1. Prediction Results

Our prediction is in the form of a confusion matrix where its column presents the prediction value. As shown in Table 1, that presents a prediction related to conflict in a test case. A pruned Classification Tree (CT) and Regression Tree (RT) outperform others. We also found that CT has only 116 (108 + 8) packages have conflicts. Consistently, the CT pruned has also 166 (99 + 17) conflict packages. Moreover, unpruned RT has double prediction of defects in their future development compares to CT that is 8 (0 + 8) for CT and 16 (16 + 10) for RT. And, for pruned RT has also almost double potential conflict than CT, that is 30 (2 + 28) for RT and 17 (0 + 17) for CT. This information gives us insight into prediction that RT outperforms CT for pruned and unpruned tree.

Table 1. Prediction on Classification Tree (CT) and Regression Tree (RT) – F(False) and T (True)

Actual	**CT**		**CT (Pruned)**		**RT**		**RT (pruned)**	
	F	**T**	**F**	**T**	**F**	**T**	**F**	**T**
False	534	0	534	0	528	6	532	2
True	108	8	99	17	106	10	88	28

Prediction using Random Forest shows a better prediction than Classification Tree and less than Regression Tree without pruned, as shown in Table 2 that predict 26 defects.

Overall, the pruned tree has enhanced the prediction of potential defects within versions of Ubuntu repositories. We also test the accuracy of our prediction by checking into Ubuntu Bug reporting system (https://bugs.launchpad.net/ubuntu/).

Table 2. Prediction using Random Forest

	CT	
	FALSE	**TRUE**
False	534	0
True	90	26

3.2. Evaluation

Performance evaluation measures the work of algorithm Machine Learning for accuracy and precision [9,10,11,12,13,14,15,16]. We use the most common method to measure performance, which are Mean Squared Error (MAE), Mean Absolute Error (MSE), and Root Mean Square Error (RMSE). Both, MAE and RMSE are sufficient to measure performance using the data set that is created using package dependency within Ubuntu repository.

Mean Absolute Error (MAE) gives equal weight to all errors that is based on the average magnitude of errors in a set of predictions beyond its direction. As shown in equation 1, where $\hat{y}_i$ is the predicted bugs and y_i is the actual bug in the repository and T is the number of examples used for testing. MAE is recommended for being unbiased towards under and overestimations and describe uniformly distributed errors.

$$MAE = \frac{1}{T}\sum_{i=1}^{T} |\hat{y}_i - y_i| \qquad (1)$$

Mean Squared Error (MSE) measures the average of the square error of the distance between prediction and actual value, as shonw in equation 2.

$$MSE = \sum_{i=1}^{T} |\hat{y}_i - y_i|^2 \qquad (2)$$

Root Mean Squared Error (RMSE) assumes of unbiased error and has normal distribution that consider between prediction and actual observation, is shown in equation 3. RMSE emphasizes large errors more and is suitable for evaluating how closed the observed data points to the predicted values.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{T} |\hat{y}_i - y_i|^2}{T}} \qquad (3)$$

As shown in Table 3, Random Forest (RF) outperforms all methods in "Accuracy" as the highest, and the lowest error rate in MAE, MSE and RMSE. For pruned RT, it has similar performance to RF that is 86.2 %. However, RT has the lowest performance compares to all models.

Table 3. Model Performance of CT = Classification Tree, RT = Regression Tree and RF = Random Forest.

Model	MAE	MSE	RMSE	Accuracy %
CT	0.2800000	1.1876923	1.0898130	83.3
CT (pruned)	0.2661538	1.1615385	1.0777469	84.8
RT	0.4430945	1.1128262	1.0549058	82.7
RT (pruned)	0.3552781	1.1015966	1.0495697	86.2
RF	0.2128813	0.926039	0.9623092	86.2

4. Conclusion

We have presented a Search-Based Software Engineering technique to predict defect in Ubuntu software repository. The dataset is obtained from Ubuntu distribution 12, 14 and 16 repositories. The training and testing is 70% and 30% of overall dataset. As the dataset is created from the dependency between packages, the data is (like) a network model. Our result shows that Random Forest model outperforms Classification Tree and Regression Tree to predict possibility of bugs in the repository with accuracy 86.2%. In our results, we found that pruned tree may produce a better result in predicting conflicts in Ubuntu repository.
For future work, we would like to investigate using fully graph method for machine learning that focuses of how a potential defect may influence functionality within Ubuntu system.

5. Acknowledgement

This work was supported by LPPM Universitas Pelita Harapan, contract number P-038-FIK/VII/2018

6. References

[1] P. Abate, R. D. Cosmo, L. Gesbert, F. L. Fessant, R. Treinen, and S. Zacchiroli. Mining component repositories for installability issues. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pages 24–33, May 2015.
[2] P. Abate and R. Di Cosmo. Adoption of academic tools in open source communities: The debian case study. Open Source Systems: Towards Robust Practices, pages 139–150, Cham, 2017. Springer International Publishing.
[3] C. Artho, R. D. Cosmo, K. Suzaki, and S. Zacchiroli.Sources of inter-package conflicts in debian. CoRR, abs/1110.1354, 2011.
[4] V. Boisselle and B. Adams. The impact of cross-distribution bug duplicates, empirical study on debian and ubuntu. In 2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM) , pages 131–140, Sept 2015.
[5] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella. Multi-objective cross-project defect prediction. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, March 2013.
[6] E. Gansner, E. Koutsofios, and S. North. Drawing graphs with dot, 2006.
[7] A. Ghannem, G. El Boussaidi, and M. Kessentini. On the use of design defect examples to detect model refactoring opportunities. Software Quality Journal, 24.4, Dec. 2016.
[8] R. Hertzog and R. Mas. The Debian Administrator's Handbook: Debian Squeeze from Discovery to Mastery, 2012.
[9] R. Malhotra, N. Pritam, and Y. Singh. On the applicability of evolutionary computation for software defect prediction. In 2014 ICACCI, Sept 2014.
[10] I. M. Murwantara, B. Bordbar, and L. L. Minku. Measuring energy consumption for web service product configuration. In Proceedings of the iiWAS'14, pages 224–228, New York, NY, USA, 2014. ACM.

[11] E. Rubini´c, G. Mauˇsa, and T. G. Grbac. Software defect classification with a variant of NSGA-II and simple voting strategies. In M. Barros and Y. Labiche, editors, Search-Based Software Engineering, 2015. Springer International Publishing.
[12] M. Sharma and A. Tondon. Developing prediction models to assist software developers and support managers. ICCSA 2017. Springer International Publishing.
[13] J. D. Strate and P. A. Laplante. A literature review of research in software defect reporting. IEEE Transactions on Reliability, 62(2):444–454, June 2013.
[14] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai. Bug characteristics in open source software. Empirical Software Engineering, 19(6):1665–1705, Dec 2014.
[15] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. The impact of automated parameter optimization on defect prediction models. IEEE Trans. on Software Engineering, 2018.
[16] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan. Cross-project defect prediction using a connectivity-based unsupervised classifier. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pages 309–320, May 2016.