Chapter 39
Improving Logging Prediction on Imbalanced Datasets: A Case Study on Open Source Java Projects

Sangeeta Lal
Jaypee Institute of Information Technology Noida, Department of CSE & IT, Noida, Uttar-Pradesh, India

Neetu Sardana
Jaypee Institute of Information Technology Noida, Department of CSE & IT, Noida, Uttar-Pradesh, India

Ashish Sureka
ABB Corporate Research Center, Bangalore, India

ABSTRACT

Logging is an important yet tough decision for OSS developers. Machine-learning models are useful in improving several steps of OSS development, including logging. Several recent studies propose machine-learning models to predict logged code construct. The prediction performances of these models are limited due to the class-imbalance problem since the number of logged code constructs is small as compared to non-logged code constructs. No previous study analyzes the class-imbalance problem for logged code construct prediction. The authors first analyze the performances of J48, RF, and SVM classifiers for catch-blocks and if-blocks logged code constructs prediction on imbalanced datasets. Second, the authors propose LogIm, an ensemble and threshold-based machine-learning model. Third, the authors evaluate the performance of LogIm on three open-source projects. On average, LogIm model improves the performance of baseline classifiers, J48, RF, and SVM, by 7.38%, 9.24%, and 4.6% for catch-blocks, and 12.11%, 14.95%, and 19.13% for if-blocks logging prediction.

DOI: 10.4018/978-1-7998-2460-2.ch039

INTRODUCTION

Debugging plays an essential role in the development of any Open Source Software (OSS), as the speed of debugging can be of vital importance in adopting any OSS. Logging, an important software development practice, is crucial for debugging in the production setting, and can play a major role in the success of any OSS. Logging is used to record execution information about the program. The recorded log assists the software developers in fixing bugs. An empirical study performed by Yuan et al. (2012a) on OSS showed that bug reports consisting of log statements are fixed 2.2 times faster than the bug report without log statements. Stack traces produced at the time of program failure are also useful in fixing the bug, but they only provide information about the exact point where the failure occurs, and do not give any information about the state just before the failure (*StackExchange*, n. d.). In contrast, log statements provide history about the failed event which is useful in debugging. In addition to debugging, logging is useful in several other software development activities, such as remote issue resolution (BlackBerry Enterprise Server Logs Submission, 2015), performance problem diagnosis (Nagaraj, Killian, & Neville, 2012), workload modeling (Sharma, Chudnovsky, Hellerstein, Rifaat, & Das, 2011), and load testing (Jiang, Hassan, Hamann, & Flora, 2008, 2009). The importance of logging can be considered from the fact that log statements are pervasive in OSS as various OSS are heavily logged. For example, the widely used OpenSSH project consist of 3407 log statements (Yuan et al. (2012b)). The Tomcat, CloudStack, and Hadoop project consists of thousands of log statements (refer to Table 10 in the Appendix).

Log statements are useful, but they have a cost-benefit tradeoff. Excess log statements in source code can generate too many trivial logs, making debugging more challenging by hiding important debugging information. Excess log statements can also increase performance (I/O intensive activity) and cost (development and maintenance) overhead. Excessive logging is one reason for poor performance in several OSS, such as Tomcat, Jetty, JBoss (Granber, 2016a; Grabner, 2016b). Like excessive logging, sparse logging is also problematic. It can omit important debugging information and decrease the benefits of logging. Hence, it is important to optimize the number of log statements in the source code.

Optimizing log statements in the source code, or identifying code constructs that must be logged, is a nontrivial and technically challenging task. It happens because software developers and code contributors in OSS often are not provided with any formal guidelines about software logging. Hence, logging is often based on domain knowledge and experience of the software developers. Also, logging practices can differ from project to project, depending on the application need. This problem can be exaggerated in OSS because contributions to OSS are often voluntary. These systems may lack documentation (Levensque, 2005) or appropriate coding standards ((Fitzgerald, 2004). All the knowledge and experience is in the mind of the experienced software developer. In addition, finding mentors in OSS is also very challenging (Steinmacher et al. (2013). As a result, source code logging can be challenging for new OSS developers who lack experience and domain knowledge. Previous studies show that software developers face difficulties in identifying source code constructs that need to be logged or in optimal source code logging (Fu, et al., 2014; Zhu et al., 2015). Hence, tools and techniques to help software developers make informed logging decisions in the source code could be beneficial.

Several recent studies propose machine-learning-based models to predict log statements in source code help software developers in identifying the source code constructs that must be logged (Fu et al., 2014; Lal & Sureka, 2016; Lal, Sardana, & Sureka, 2016; Saini, Sardana, & Lal, 2016; Zhu et al., 2015). These techniques use static features from the source code to train the machine-learning-based model used to predict logged and nonlogged source code constructs. However, a major challenge in machine-