Stack Trace 기반 Bug report 우선순위 자동 추천 접근 방안

이정훈*, 김태영*, 최지원*, 김순태*, 류덕산*
*전북대학교 소프트웨어공학과
jhmake4419@gmail.com, rlaxodud1200@jbnu.ac.kr, jiwon.choi@jbnu.ac.kr, stkim@jbnu.ac.kr, dsryu99@gmail.com

An Automatic Approach for the Recommendation of Bug Report Priority Based on the Stack Trace

JeongHoon Lee*, Taeyoung kim*, Jiwon Choi*,SunTae Kim*, Duksan Ryu*
*Dept. of Software Engineering, Jeon-Buk National University

요 약

소프트웨어 개발 환경이 빠르게 변화함에 따라 시스템의 복잡성이 증가하고 있다. 이에 따라 크고 작은 소프트웨어의 버그를 피할 수 없게 되며 이를 효율적으로 처리하기 위해 Bug report 를 사용한다. 하지만, Bug report 에서 개발자가 해당 Bug report 의 우선순위를 결정하는 과정은 노력과 비용 그리고 시간을 많이 소모하게 만든다. 따라서, 본 논문에서는 Bug report 내의 Stack trace 를 기반으로 Bug 의 우선순위를 자동적으로 추천하는 기법을 제안한다. 이를 위해 본 연구에서는 첫 번째로 Bug report 로부터 Stack trace 를 추출하였으며 Stack trace 의 3 가지 요소(Exception, Reason 그리고 Stack frame)에 TF-IDF, Word2Vec 그리고 Stack overflow 를 사용하여 특징 벡터를 정의하였다. 그리고 Bug 의 우선순위 추천 모델을 생성하기 위해 4 가지의 Classification 알고리즘을(Random Forest, Decision Tree, XGBoost, SVM)을 적용하였다. 평가에서는 266,292 개의 JDK library 의 Bug report 데이터를 수집하였고 그중 Stack trace 를 가진 Bug report 로부터 68%의 정확도를 산출하였다.

1. 서론

최근 급변하는 소프트웨어 분야 속에서 소프트웨어 개발환경 또한 빠르게 변화하고 있다. 이에 따라 소프트웨어의 규모와 복잡성이 증가하고 있으며 소프트웨어의 크고 작은 버그를 피할 수 없게 되었다. 이러한 버그들을 다루기 위해 개발자들은 Bug tracking system(Windows Error Report tool[1], Mozilla crash reporting system [2], Ubuntu's Apport crash reporting tool [3], etc.) 으로부터 사용자들에게 버그리포트를 받고 해당 Bug report 의 내용, 개발자 본인의 경험 그리고 과거의 Bug report 들을 확인하여 버그의 우선순위를 결정한다. 하지만, 이러한 과정은 개발자의 노력과 비용 그리고 시간을 많이 소모하게 만든다.

사용자들은 버그 리포트를 제출 시 버그에 대한 텍스트 형식의 설명과 Stack trace 등을 포함한 다양한 정보를 기입한다. 이 중 Stack trace 는 SW 프로젝트의 충돌 또는 결함이 발생했을 때 Memory stack 안에 생성되는 Function Call 들의 순서 집합으로써 다음 그림 1)과 같이 Exception, Reason 그리고 Stack frame 으로 구성된다.

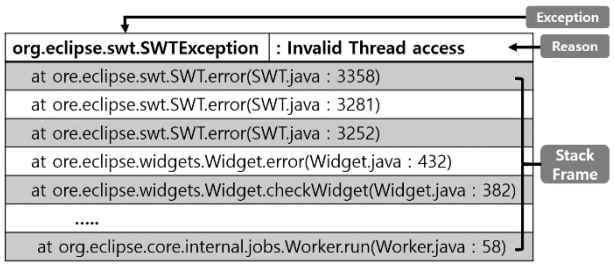

그림 1) Stack trace 의 구조

이러한 3 가지 정보를 가지는 Stack trace 는 개발자가 해당 버그의 우선순위를 판단하는데 중요한 역할을 한다. 먼저, Stack trace 의 Exception 으로부터 개발자는

[1] http://msdn.microsoft.com/en-us/library/windows/hardware/gg487440.aspx
[2] http://crash-stats.mozilla.com
[3] https://wiki.ubuntu.com/Apport

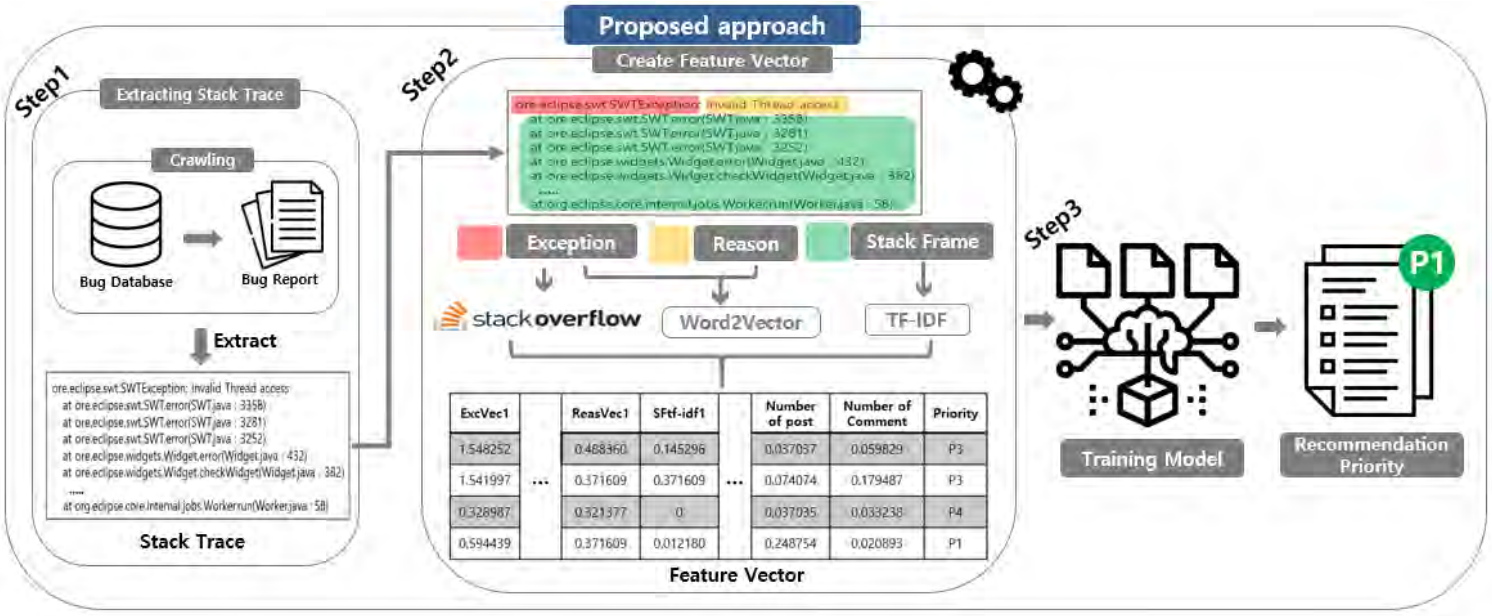

그림 2) Proposed approach

시스템으로부터 어떠한 에러가 발생했는지 확인하며 Reason 으로부터 해당 Exception 이 발생되는 이유를 파악하게 된다. 이러한 내용을 기반으로 Stack frame 에 있는 Function call 들로부터 버그가 발생할 수 있는 파일들을 확인 후 각 파일을 들어가서 버그를 수정하게 된다.

이러한 Stack trace 를 활용하여 버그리포트의 우선순위를 예측하는 기존의 연구들이 존재한다. 일반적인 버그의 우선순위 추천 연구들은 자연어 처리 방식을 Bug report 의 Stack trace 를 제외한 Description 의 정보를 적용하여 버그의 우선순위를 추천한다[1]. 하지만 이러한 방식들은 Bug 에 대한 시스템 적인 내용을 충분히 가지고 있지 않고 있으며 Stack trace 를 활용한 연구들의 경우 Stack trace 에서 Stack frame 만의 정보를 TF-IDF 를 사용하여 Bug 우선순위를 추천해주고 있다[2][3]. 하지만 이러한 방식은 Stack trace 에 존재하는 다른 요소들(Exception 그리고 Reason)을 전혀 반영하고 있지 않기 때문에 버그 우선순위를 추천하는데 중요한 정보를 놓칠 수도 있다.

따라서, 본 연구는 이러한 문제에 접근하기 위해 Stack trace 를 기반으로 Bug report 우선순위 추천 방안에 대해 제안한다. 첫 번째로, Bug report 들로부터 Stack trace 들을 추출하였으며, Stack trace 의 3 가지 요소(Exception, Reason, Stack frame)로부터 특징벡터를 생성하기 위해 Word2vec, TF-IDF 그리고 Stack Overflow 의 글에 대한 정보를 활용한다. 생성된 특징벡터를 기반으로 여러 Classification 머신 러닝 알고리즘들을 적용하여 버그 심각도 추천 모델을 생성한다. 본 연구의 접근방안의 적절성과 버그리포트의 우선순위 예측 모델의 성능을 평가하기 위해 평가단계에서 2 가지 Research question 을 선정하였으며 실험을 통해 68%의 정확도를 도출하였다.

제안하는 기법의 설명을 위해 본 연구는 다음과 같이 구성된다. 2 장에서는 Stack trace 를 사용한 버그리포트 우선순위 추천에 관한 기존연구들을 분석한다. 3 장에서는 Stack trace 기반의 버그리포트 우선순위 추천 방안에 대해 제안한다. 4 장에서는 제안하는 접근방법의 적절성과 모델의 성능에 관해 기술하였고 5 장에서 결론을 맺는다.

2. 접근 방법

이 장에서는 Stack trace 기반의 버그 우선순위 예측을 위한 접근방안에 대해 소개한다. 그림 2)에서와 같이 우리의 접근방안은 크게 3 가지 단계로 구성된다. 첫 번째로 Bug report 의 Description 에서 Stack trace 들에 대해 추출하였으며 두 번째에서는 추출된 Stack trace 를 기반으로 Feature vector 들을 생성한다. 마지막으로 생성된 Feature vector 로부터 Classification 모델을 적용하여 Bug 우선순위 추천 모델을 생성한다.

2.1. Extracting Stack Trace

이 단계의 주요 목표는 Stack trace 기반의 특징벡터를 생성하기 위해 Bug report 들로부터 Stack trace 들을 추출하는 것이다. 다음의 그림 3 은 Bug report 의 구조를 보여주고 있다.

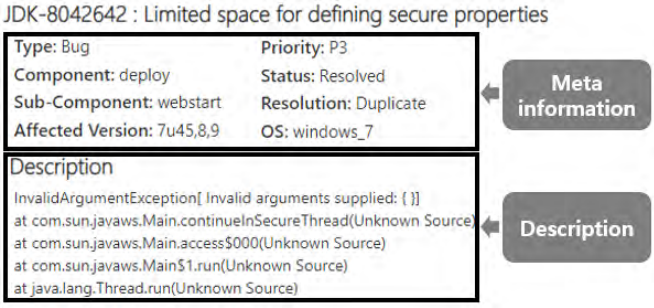

그림 3) 버그리포트의 구성

버그 리포트는 크게 Meta information 과 description 2 가지 구조로 구성되어 있다. 첫 번째로, Meta information 에서는 해당 버그리포트의 제출일, 상태, 개발 환경 등의 meta 정보들을 보여주고 있으며, description 에서는 해당 버그에 대한 설명, Stack trace 의 내용들 담고 있다. 즉, 우리는 Bug report 의 description 을 기반으로 하여 Stack trace 를 추출한다[4][5].

3.2. Creating Feature Vector

이 단계에서는 다음의 그림 4)와 같이 추출된 Stack trace 를 기반으로 Word2Vec, TF-IDF 그리고 Stack overflow 를 사용하여 3 가지 유형의 특징벡터를 생성한다. 그 다음 각 유형의 특징벡터를 결합하여 Bug 우선순위 추천 모델을 생성하기 위한 특징벡터를 구성한다.

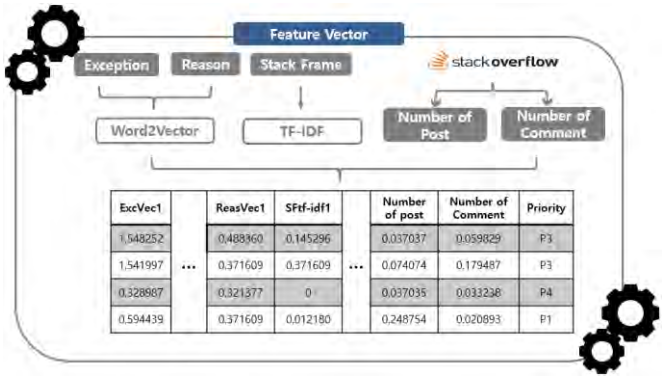

그림 4) Creating Feature Vector

3.2.1. Word2Vec Based Feature Vector

Exception 과 Reason 에 대한 의미적 정보를 특징화하기 위하여 우리는 Word2Vec[6]을 적용한다. Word2Vec 은 단어들의 의미적 차이를 구별하고 컴퓨터가 이해하도록 해주는 대표적인 Word Embedding 기법이다. 우리는 각 Bug report 의 Stack trace 에 존재하는 모든 Exception 과 Reason 에 대해 각각의 Word2Vec Model 을 구성하고 이를 기반으로 각 Exception 과 Reason 을 벡터로 변환한다.

3.2.2. TF-IDF Based Feature Vector

TF-IDF[7] 기반의 특징벡터는 Stack frame 의 function call 들을 기반으로 생성된다. TF-IDF 는 특정 문서내의 단어가 여러 문서에 얼마나 자주 등장하는지를 값으로 변환하는 통계적 기법으로 그 값을 통해 해당 단어가 문서내에 중요한 내용임을 가리키는 지표가 된다. 즉 우리는 Stack frame 에서 중요한 function call 들의 내용들에 대한 값들을 표현하기 위해 Sabor[2]의 연구에서 사용된 TF-IDF 기반의 Stack frame 특징벡터를 사용한다.

3.2.3. Stack Overflow based Feature Vector

Stack overflow 기반의 특징 벡터를 생성하기 위해, 우리는 Stack trace 의 Exception 을 사용하여 Stack overflow API[8]를 통해 검색된 글의 수와 각 글의 Comment 수를 얻으며 이를 평균값을 내어 특징벡터로 생성한다. Stack Overflow 에서 Bug report 에서 발생한 Exception 을 사용하는 이유는 많은 개발자들이 버그를 수정할 때에 Stack overflow 의 정보를 많이 활용하며 이때, 해당 Exception 이 많이 검색되는 경우에는 쉽게 버그를 해결할 수 있어 Bug 의 우선순위가 낮아질 수 있기 때문이다.

마지막으로 3 가지 유형에 따라 생성된 특징벡터는 하나의 특징벡터로 결합되어 지며 Number of Post, Number of comment 에 대해서는 min-max normalization 을 적용하였다. 마지막에 Bug 우선순위에 대한 Label 이 끝에 붙여진다.

3.3. Creating Train Model

이전 단계에서 생성된 특징 벡터들은 Bug 의 우선순위를 추천해주기 위한 모델을 생성하기 위해 사용된다. 그리고 생성된 모델은 새로 들어온 Bug report 에 대해서 Bug 의 우선순위를 추천해준다. Bug 우선순위 추천 모델은 Random Forest, Decision Tree, XGBoost 그리고 SVM 과 같은 Classification 알고리즘을 적용하여 생성하며 각 알고리즘은 Skit-learn[9]을 통해서 적용하였다.

4. 평가

본 연구의 접근방안의 적절성과 버그리포트의 우선순위 예측 모델의 성능을 평가하기 위해 평가단계에서 다음의 2 가지 Research Question 을 선정하였다.

- RQ 1. : 정의한 특징 벡터가 버그리포트의 우선순위를 예측하기에 적절한가?
- RQ 2. : 얼마나 훈련된 모델이 버그리포트의 우선순위를 잘 추천하는가?

실험 데이터

평가를 수행하기 위해 우리는 우선 Oracle java Bug Database 에 제출된 JDK(Java Development Kit) [10] 라이브러리의 Bug report 를 수집하였다. JDK 는 자바 언어 기반의 SW 를 개발하기 위해 사용되는 Library 이며 현재 까지도 유지보수를 잘 수행하고 있는 프로젝트이기 때문에 실험 데이터 셋으로 선정하였다. Table 1 과 같이 우리는 총 266,926 개의를 수집하였고 그 중 7,273 개의 Stack trace 가 포함된 버그리포트를 사용하였다.

	Total bug report	Bug report with stack trace
Bug report	266,926	7,273

Table 1) the number of bug report

RQ. 1 정의한 특징 벡터가 버그리포트의 우선순위를 예측하기에 적절한가?

첫 번째 질문에 답하기 위하여 scikit-learn 을 사용하여 다중 회귀 분석을 적용하였다. 이 질문에서의 목적은 우리가 정의한 특징벡터가 버그리포트의 우선순위 추천에 얼마나 영향력을 보기 위함이기 때문에 다음의 Table 2. 와 같이 TF-IDF 를 적용하기 전과 후로 구분하여 R-square 와 Adjust R-square 값을 도출하였다.

TF-IDF 를 적용하였을 때는 0.974 와 0.920 의 값을 얻었으며 TF-IDF 를 미적용 한 경우에는 0.825 와 0.816 의 값을 얻었다. 이를 통해 TF-IDF 를 제외한 우리의 정의된 특징 벡터만으로도 모델을 잘 설명한다는 것을 보여주고 있으며 TF-IDF 를 적용 시 모델의 설명력이 더욱 올라감을 보여준다.

	TF-IDF 적용	TF-IDF 미적용
R-squared	0.974	0.825
Adj.R-squared	0.920	0.816

Table 2) 다중 회귀 분석 결과

RQ. 2 훈련된 모델이 버그리포트의 우선순위를 잘 예측하는가?

두 번째 질문은 우리의 훈련된 모델이 얼마나 잘 Bug 의 우선순위를 추천해 주는지 알아보기 위함이다. 즉, 우리 모델의 성능을 평가하기 위해 우리는 Data set 을 Training set 과 Validation set 그리고 Test set 3 가지로 나눈 후 모델에 대해 평가하였다. 또한, Validation set 과 Test set 은 전체 Data set 에서 무작위로 선출하였다. 다음의 Table 3. 은 4 개의 Classification 알고리즘을 적용하여 나온 성능의 결과를 보여주고 있다. 결과에서 우리의 모델은 평균적으로 68%의 정확도를 가지고 있으며 또한 Random forest 에서는 75%의 정확도를 가짐을 보여주고 있다.

	Precision	Recall	F-measure	Accuracy
Random Forest	0.712	0.562	0.604	0.75
Decision Tree	0.554	0.540	0.548	0.71
SVM	0.494	0.502	0.488	0.65
XGBoost	0.454	0.408	0.406	0.62
Average	0.5535	0.503	0.5115	0.68

Table 3) 다중 분류 모델 적용 결과

5. 결론

본 연구는 개발자의 버그리포트 우선순위 결정 시 Stack trace 를 기반으로 우선순위를 예측, 추천하는 방안을 제안했다. 먼저, Word2Vec 과 TF-IDF 그리고 Stack overflow 를 활용하여 Stack trace 로부터 특징벡터를 생성한 후 Random Forest, Decision Tree, SVM, XGBoost 등의 Classification 알고리즘을 적용하여 모델을 생성하였다. 그 후, 우리의 생성된 모델과 특징벡터의 유의미함을 평가하기 위해 다중회귀와 모델의 성능 평가를 진행하였고 평균적으로 68%의 정확도를 얻어 내었다. 향후 연구로는 Stack Trace 뿐만 아니라 버그리포트의 Meta information 의 내용들을 특징 벡터로 추가할 예정이며 Stack frame 안의 Function call 들의 순차적 특성을 RNN 에 적용하여 새로운 특징 벡터를 추가 정의하여 성능 향상을 할 예정이다.

Acknowledgments

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임.
(NO.2020R1F1A1072039)

참고 문헌

[1]Lamkanfi, Ahmed, et al. "Comparing mining algorithms for predicting the severity of a reported bug." 2011 15th European Conference on Software Maintenance and Reengineering. IEEE, 2011.

[2] Sabor, Korosh Koochekian, Mohammad Hamdaqa, and Abdelwahab Hamou-Lhadj. "Automatic prediction of the severity of bugs using stack traces and categorical features." Information and Software Technology 123 (2020): 106205.

[3]Sabor, Korosh Koochekian, Mohammad Hamdaqa, and Abdelwahab Hamou-Lhadj. "Automatic prediction of the severity of bugs using stack traces." Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering. 2016.

[4]Bettenburg, Nicolas, et al. "Extracting structural information from bug reports." Proceedings of the 2008 international working conference on Mining software repositories. 2008.

[5]https://github.com/kuyio/infozilla

[6]Goldberg, Yoav & Levy, Omer. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method.

[7]Rajaraman, A., & Ullman, J. (2011). Data Mining. In Mining of Massive Datasets (pp. 1-17). Cambridge: Cambridge University Press. doi:10.1017/CBO9781139058452.002

[8]https://api.stackexchange.com/

[9]https://scikit-learn.org/stable/

[10]https://www.oracle.com/java/technologies/java-se-glance.html