Feature Selection and Embedding Based Cross Project Framework for Identifying Crashing Fault Residence

Zhou Xu[a,b], Tao Zhang[c], Jacky Keung[d], Meng Yan[a,*], Xiapu Luo[b,*], Xiaohong Zhang[a], Ling Xu[a]

[a] *School of Big Data and Software Engineering, Chongqing University, Chongqing, China*
[b] *Department of Computing, The Hong Kong Polytechnic University, Hong Kong*
[c] *Faculty of Information and Technology, Macau University of Science and Technology, Macao, China*
[d] *Department of Computer Science, City University of Hong Kong, Hong Kong, China*

1. Experimental Setup

Independent Variable Extraction

In order to characterize each crash instance (i.e., crashing fault), Gu et al. [1] extracted 89 features as the independent variables in total from the stack trace and the relevant source code. These features come from 5 families as follows:

- 11 features based on the stack trace (marking from ST01 to ST11 for simplicity). These features represent the difficulty of handling the corresponding crash.
- 23 features based on the function and class in the top frame (marking from CT01 to CT23). As the position in which the exception is thrown is located in the top frame, the features of the function and class in the top frame can characterize the program state when it encounters a crash.
- 23 features based on the function and class in the bottom frame (marking from CB01 to CB23). As the information of initial function call is recorded in the bottom frame, the feature of the function and class in the bottom frame can also characterize the crashing fault.

*The corresponding authors.
Email addresses: mengy@cqu.edu.cn (Meng Yan), csxluo@comp.polyu.edu.hk (Xiapu Luo)

- 16 features by normalizing CT08∼CT23 with LOC (marking from AT01 to AT16).
- 16 features by normalizing CB08∼CB23 with LOC (marking from AB01 to AB16).

Their brief descriptions are summarized in Table 1.

2. Discussion

In this section, we discuss how different feature dimensions and parameter settings impact our experimental results.

2.1. The Impact of the Selected Feature Dimensions on the ICFR Performance of FSE Framework

After the feature embedding with WBDA, the original cross project data are mapped into a new space. In our experiment, we conduct experiments with the dimension of the reserved feature as 15% of the original feature number which was suggested in [2, 3]. In this subsection, we discuss the impacts of different feature dimensions on the cross project ICFR performance of FSE framework. We set 20 feature dimensions, from 5% to 100% with a step of 5%, for observation. Figure 1 depicts the 6 average indicator values across 42 cross project pairs of our proposed FSE framework under 20 different feature dimensions.

From this figure, we observe that the performance of FSE framework with feature dimensions of 5%, 10%, and 15% is higher than that with other dimensions in terms of F(InTrace), g-mean, Balance, MCC, and AUC, whereas the phenomenon is reversed in terms of F(OutTrace). In addition, the performance of FSE framework with dimensions larger than 20% keeps stable.

To sum up, FSE framework obtains the better performance with smaller feature dimensions but similar performance with larger feature dimensions on 5 indicators. Overall, the dimensions between 5% and 15% can be more suitable for FSE framework.

Table 1: Brief Descriptions of 89 Features for Crash Instances

Feature	Description	
Set CT and CB	**Features related to the top frame CT (bottom frame CB)**	
CT01 (CB01)	in top (bottom) class	Number of local variables
CT02 (CB02)		Number of fileds number
CT03 (CB03)		Function (except constructor one) number
CT04 (CB04)		Imported packages number
CT05 (CB05)		Whether the class is inherited from others
CT06 (CB06)		LoC of comments
CT07 (CB07)	in top (bottom) function	LoC
CT08 (CB08)		Number of parameters
CT09 (CB09)		Number of local variable
CT10 (CB10)		Number of if-statements
CT11 (CB11)		Number of loops
CT12 (CB12)		Number of for statements
CT13 (CB13)		Number of for-each statement
CT14 (CB14)		Number of while statements
CT15 (CB15)		Number of do-while statements
CT16 (CB16)		Number of try blocks
CT17 (CB17)		Number of catch block
CT18 (CB18)		Number of finally blocks
CT19 (CB19)		Number of assigment statements
CT20 (CB20)		Number of function calls
CT21 (CB21)		Number of return statements
CT22 (CB22)		Number of unary operators
CT23 (CB23)		Number of binary operators
Feature Set ST	**Features related to the stack trace (short for ST)**	
ST01	Type of the exception in the crash	
ST02	Number of frames of the ST	
ST03	Number of classes of the ST	
ST04	Number of functions of the ST	
ST05	Whether an overloaded function exists in ST	
ST06	Length of the name in the top class	
ST07	Length of the name in the top function	
ST08	Length of the name in the bottom class	
ST09	Length of the name in the bottom function	
ST10	Number of Java files in the project	
ST11	Number of classes in the project	
Feature Set AT	**Features normalized by LoC**	
AT01 (AB01)	CT08/CT07 (CB08/CB07)	
AT02 (AB02)	CT09/CT07 (CB09/CB07)	
AT15 (AB15)	CT22/CT07 (CB22/CB07)	
AT16 (AB16)	CT23/CT07 (CB23/CB07)	

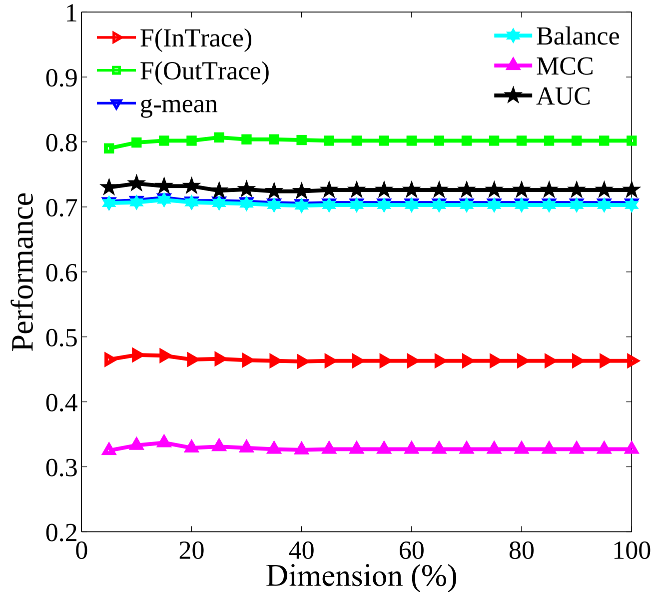

Figure 1: The 6 average indicator values of FSE framework under different feature dimensions.

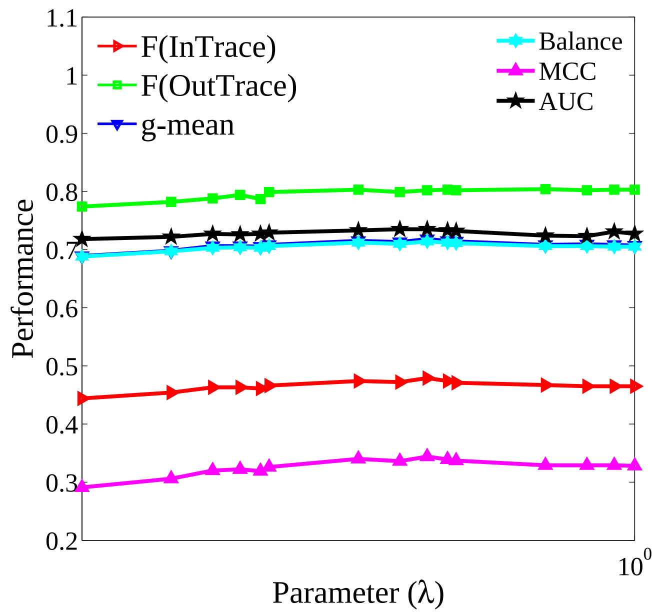

Figure 2: The 6 average indicator values of FSE framework with different parameter λ values.

2.2. The Impact of Parameter λ Values on the ICFR Performance of FSE Framework

In this work, we conduct experiments by setting the regularization parameter λ value as 0.1 without any prior knowledge. In this subsection, we discuss the impacts of different parameter λ values on the cross project ICFR performance of FSE framework. We set 15 parameter λ values, from 0.001 to 0.009 with a step of 0.002, 0.01 to 0.09 with a step of 0.02, and 0.1 to 0.9 with a step of 0.2 for observation. Figure 2 depicts the 6 average indicator values across 42 cross project pairs of our proposed FSE framework with 15 λ values.

From this figure, we observe that the performance of FSE framework with λ values larger than 0.01 is similar and higher than that with other λ values in terms of all indicators.

Overall, FSE framework obtains the better and stable performance with larger λ values and the λ values between 0.03% and 0.9% can be better choices for FSE framework.

References

[1] Y. Gu, J. Xuan, H. Zhang, L. Zhang, Q. Fan, X. Xie, T. Qian, Does the fault reside in a stack trace? assisting crash localization by predicting crashing fault residence, Journal of Systems and Software (JSS) 148 (2019) 88–104.

[2] S. Shivaji, E. J. Whitehead Jr, R. Akella, S. Kim, Reducing features to improve bug prediction, in: 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE, 2009, pp. 600–604.

[3] S. Shivaji, E. J. Whitehead, R. Akella, S. Kim, Reducing features to improve code change-based bug prediction, IEEE Transactions on Software Engineering (TSE) 39 (4) (2012) 552–569.