Rank and Classification of Bug Reports and Feature Evaluation Using SVM

Gomathi S¹, Swathini K², Suvetha M³, Deepa M⁴

¹Assistant Professor, Department of Computer Science and Engineering, Sri Krishna College of Technology, Coimbatore

^{2, 3, 4} UG Scholor, Department of Computer Science and Engineering, Sri Krishna College of Technology, Coimbatore

Abstract: When a new bug report is received, developers usually need to procreate the bug and perform code reviews to find the origination, this process can be tedious and time consuming. A instrument for ranking all the source files with regards to how likely they are to contain the cause of the bug would enable developers to narrow down their search and better result. This paper introduce an adaptive ranking approach that leverages project acquaintance through functional decomposition of source code, API metaphors of library components, the bug-fixing record, the code change record, and the file dependency graph. Given a bug report, the ranking evaluation of each source file is computed as a weighted collection of an array of feature film, where the weights are trained mechanically on previously resolved bug reports using a learning-to-rank technique. We estimate the ranking system on six large scale open source Java projects, using the earlier-fix version of the project for every bug report. The explore results show the Support Vector Machine approach that outperforms in defect prediction.

Keywords: SVM, Bug Localization, Ranking, Information Retrieval

1. Introduction

Software is more than just a program code. A program is an executable code, which suffice some computational reason. Software is measured to be collection of executable programming code, related libraries and documentations. Software, when made for a particular necessity is called software product. A software bug is a coding mistake that may cause an unplanned on unexpected action of the software component. Upon discovering an uneven manners of the software project, document will be reported by developer or user, called a bug report. A bug report supply information that could help in repair a bug, with the overall aim of improving the software quality. A ample number of bug reports could be opened during the development lifecycle of a software product. In a software team, bug reports are extensively used by both managers and developers in their daily development process.

A developer who is appointed a bug report usually needs to reproduce the irregular behavior and perform code reviews in order to find the cause. However, the variety and uneven quality of bug reports can make this process nontrivial. Essential information is often missing from a bug report. If the bug report is understand as a query and the source code files in the software repository are viewed as a group of documents, then the problem of discovering source files that are relevant for a given bug report can be formed as a standard task in information retrieval (IR).

2. Literature Review

Locating bugs is essential, hard, and high-priced, particularly for large-scale environments. To address this, natural language information retrieval techniques are increasingly being used to propose potential faulty source files given bug reports. Our key insight is that structured information retrieval based on code constructs like class and method names, enables more surgical bug localization. We present BLUiR (Bug Localization Using information Retrieval), which embodies this insight, requires only the source code and bug reports, and takes benefit of bug similarity data iflavailable.

Developers often learn to use APIs (Application Programming Interfaces) by focusing at existent examples of API usage. Code repositories contains instances of such usage of APIs. Nevertheless, usual information retrieval techniques fail to do well in retrieving API usage examples from code repositories. This paper presents SSI(Structural Semantic Indexing), which is a technique to associate words to source code entities based on sameness of API usage. The formula following this technique is that entities (classes, methods, etc.) that show parallel uses of APIs are semantically related because they do similar things.

3. Mapping Bug Reports to Relevant Files

We posture to approach it as a ranking problem, in which the source files are ranked with respect to their relevance to a given bug report. Here, relevance is equated with the likeliness that a particular source file contains the basis of the bug represented in the bug report. The ranking task is defined as a weighted combination of features, where the features depict to a great extent on knowledge specific to the software engineering domain in order to assess relevant relation among the bug reports and the source code files. While a bug report may share textual tokens with its relevant source files, in generic there is a significant inherent counterpart between the natural language hired in the bug report and the programming language used in the code. We initiate a learning-to-rank approach that emulates the bug finding process employed by developers. The ranking model qualify useful relationships between a bug report and source code files by leveraging domain acquaintance, such as API specifications, the syntactic structure of code, or issue tracking data The adaptive ranking approach is commonly applicable to software projects for which there exists a

adequate amount of project specific knowledge, such as a comprehensive API documentation and an initial number of previously fixed bug reports. To determine a bug, developers use not only the content of the bug report but also domain knowledge relevant to the software project. Furthermore, the proposed ranking model exceed three recent state-of-the-art approaches. Feature selection experiments employing greedy backward feature eliminations, it shows that all features are useful. When united with runtime analysis, the feature selection results can be utilized to pick a subset of features in order to conquer a target trade-off between system accuracy and runtime complexity. Furthermore, the ranking performance can benefit from instructive bug reports and well documented code major to a better lexical similarity, and from source code files that already have a bug-fixing history. Correspondence the bugs to the relevant files is a paper based documents, it did not have the clear idea about the bug. This system uses only less ranking models. It consumes large amount of time for reproduce the problem.

4. Support Vector Machine (SVM)

In machine learning, support vector machines (SVMs, also support vector networks[4]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression investigation. Given a set of training examples, each noticeable as belonging to one on the other of two categories, an Support Vector Machine training algorithm constructs a model that allocates new examples to one collection or the other, building it a non-probabilistic binary linear classifier. An SVM model is a sign of the examples as points in space, plotted so that the examples of the separate collection are divided by a clear gap that is as wide as possible.In our work, we will leveraging further types of domain knowledge, such as the stack traces submitted with bug reports and the file change history, as well as features previously used in fault prediction systems. We also plan to use the ranking SVM with nonlinear kernels and further measure the approach on projects in other programming languages. Using Support Vector Machine algorithm we can easily find out the bug and it can be quickly employed by the developers. This system uses more ranking methods and it takes less time for processing.

When data are not labeled, supervised learning is not probable, and an unsupervised learning approach is required, which attempts to find earthy clustering of the data to groups, and then plot new data to these groups. The clustering algorithm provides an betterment to the SVM is recognized as support vector clustering[2] and is repeatedly used in industrial applications either when data are not labeled or when only some data are labeled as a preprocessing for a classification pass.

4.1 Preprocessing

The first step towards managing and inspect textual data formats in general is to consider the text based information available in free formatted text documents or text. Initially the pre-processing is done by the following process. *Removing Stop words and Stem words*[3]. The first step is to remove the un-necessary information available in the sentence of stop words. These include few verbs, conjunctions, disjunctions and pronouns, etc. (e.g. is, am, the, of, an, we, our) and Stemming words e.g. 'deliver', 'delivering' and 'delivered' are stemmed to 'deliver'.

4.2 Collaborative Filtering

In collaborative filtering process, if previously fixed bug reports are textually similar with the current bug report, then the files that have been associated with the similar reports may also be relevant for the current report[6]. It has been observed in that a file that has been fixed before may be responsible for similar bugs. The feature computes the textual similarity between the text of the current bug report and the summaries of all the bug reports there is not much historical information that can be used for computing features that are based on collaborative filtering or the file revisal history. In particular, there is fewer possibility for exploiting duplicated bug reports.

4.3 Bug Report Analysis

In general there are seven types of bugs are accessible, Functionality Errors, Communication Errors, Missing command errors, Syntactic Error, Error handling errors, Calculation Errors and Control flow errors. The bug reports can be analyzed for which kind of bugs[5].

4.4 Ranking

The resultant ranking function is a linear collection of features, whose weights are automatically trained on previously solved bug reports using a learning-to-rank technique[8]. The ranking approach to the problem of mapping source files to bug reports that enables the unseamed integration of a wide diversity of features; exploiting before fixed bug reports as training examples for the planned ranking model in conjunction with a learning-to-rank technique.

4.5 Classification Result

In this classification, the bug reports has been analyzed and classified into which categories[4]. The model of the training task as a classification in which bug reports and files are assigned to multiple topics, we directly train our framework for ranking, which we believe is a better match for the way the model is used.

					the be	-						
1.5	-	in an		44.7	Long.	(and and	lan.m	Sec. Summe	in.	in.	- 10	
100			1	40407	ing statistics		ALC: NOT THE	Inches	and services	and a	100	
		and the		400.4	Real Property lies	Transfill goal.			insering.	0.000		
		-		41075	Ballin'nar	to defend	2010/11/11	101000	induction in the	Longe-		
1.2		interior.	18	1100	Desired.	deriver.	0100-011	178000	and shot that	14164	_	
		-		410108	the left filling		0100410		-	1004		
		-		1004	No. Other Land	And and the other	where her	100000	-	instant 1		
		-		410.0	Aprillian.	Anna Inches	Madesard.	(Passing)	and the second	10.4	_	
		interim.		4007	August for the	Property at	and so the	101003	manufact in	statio -	_	
		10000		4810	Barble and		anished on the	- manual -	and the	44.10		
		incire."	14		the deside on	agence of	10100-004		instanting	nine		
		1000.00		40584	National International Interna		investigation.	-	instant but	4007		
		1000100	10	140.0	by lotting.	Anna and		10000	instanting.	unality.	_	
		-	14	01104	Aug. 101 (101 - 102)	-	010108-011		-	100100		
		-		14110	Augustine.		100000-000	1000000	and in the local	4975		
		Antipa .		testin-	Aug. House Star		A	(reduced)				
		land an		10000	84,0000-011	And in case of	ALC: U.S. 199	1044000	-	1000	_	
		-	14	1000	Na SHOT IN	10.000 0114	and a feature	-mention	minine.	All all a		
		1000.00		100.01	ing Well-one	for the state of	manual sta	-040000	instanting .	and a		
		Name and Address		(min)	the million	the stated	10101-01404	10,000	-mainting	10400		
		inert de	14	10-10	No. OF COM.		ACC 1998 1911	140400	cond-basi	410.00		
		-			04 000 mile	The section sect.	*****	141.000	instanted.	444.0		
		terms.		1000	14,000,000		10000	10/1002	manufact.	10001	- 10	
		terio de	10	100752	10x 1070 mil	1015-00-00	State 7 to a	14/1403	manufact in	and a		
	-							_	_	_		

Figure 1: Bug Reports(Dataset)

Volume 6 Issue 2, February 2017 <u>www.ijsr.net</u> Licensed Under Creative Commons Attribution CC BY

		_							-	-	
		1	-	-							
-	10.00		Ac.7	lower .	laune .	Part In	fact, here	en like	last.		
	1000	1	ALC: N	Autorit.		anvine to	100205	induction in	ARM	1	
11	April 10	1.1	4001	As ODVIA	Indiana	204244	10,000	insisting.	1254.0		
	Aprel de	1	450	ALCOINT:	To feels at	20101042	10,000	industrial	N/r/h		
	April Do		100	hartist?	the last	204415	174000	terrar had	10.00		
	April de	1	49.08	Articles		20082141	100,000	instanting.	2084		
	April da		1004	Ra Dillional	Antonio			miniful	and .		
	familia.		4101	Aprillion.	-		1.00	animited.	2016		
	And in		4707	April 10	1.00		- 6	and set that	wate		
	April de	- 6-	4671	Aprilli and		the state of the s	- 6	-	Fig.10		
	dant de	14	-0405	by million.	1.0			interfact	All N		
	April da	12	0.070	by states.	- T			industrial.	1007		
	And in	1	1923	bermin's	1	(4)	_ E-	mental	point and		
	April No.	1	10.00		1		THE OWNER.	and the second	white		
	-	1	100	Ag Stillion		301016	104004	and the	403		
	And in	-	100	As we have		20010141	104203	and the	-		
		-			-						
	-		3448		(PRO) and		194604	and the	4940		
	April de			A			-	- and the l	Sec.		
	April No.	. 8	3427	\$4.30° m	The first ship		040035	- and that	201		
	-		3628	M NOT	Berndy	2010.081	10,9622	minihal	MAD		
	e Theighter T	e		e I Figu	and the second se	2: Pro	epro	ocess	sing	 	
			forth Report	Figu	re 2	2: Pro			lanery Nam		
			forth Honoral	Figu	re 2	2: Pro	Bar And And		lance from		
-		-	forth Report	Figu	re 2	2: Pro		. 2	lanery Nam	l Inter No. Internation	
-		-	facts Reserved	Figu	re 2	2: Pro	ter and has	. 2	lance from		
	-		Forth Deserval	Figu	re 2	2: Pro	1001	is the	100		
		-	facts Reserved	Figu	re 2	2: Pro	Right 1	11111			
	-	and and	Furth Reserval	Figu	re 2	2: Pro	IRIGHT 1	UISIN -		Nill,	
	-	and a second	Forth Reserved Inc. Down Inc. D	Figu	re 2	2: Pro	100001			,IHW	
			Furth Reserval	Figu	ire 2	2: Pro	1 10000	THE STATE		Nill,	
			Furth Downsal No.7 Second Se	Figu	ire 2	2: Pro	100001	THE PARTY IN THE PARTY INTERPARTY	amme I	- IBINI	
		and a second sec	Tenth Houseral Inc. 1 Jones 1 No.000 10 Antes 10	Figu	ire 2		1 1111111	(HIIIII)	Namus	IIIMII,	



Figure 3:Stop words removal

4	-	-	tanha:		140					
-	and the second second	<u></u>		1 HOL TUDA						
-	-	-		1007040						
-	- C	(I BUTWIND					- 1	
÷	-			10,100,000		-	a			20107
5	-	1		1276621026		-			- 1	
	-	5		100000-0-0		15	A COLOR			
	ine .	1		10100027					1	100
3	attest	1		I STREET		1.1	Water I.		- 1	States and
2	ration	1	1.000	198969.	DALLING.	-	-			Access Name
3	-	1	1.000	LANSING.	Astronomic .		6.3	A	- 1	
3	-) . · · ·		101000	1. Million State	1000		1		
3	-) bissesses	11,0,01014	LOWING.		A			
3	(And and	3		1 PECKER	A WOMPONT.	1.00	1 A C	- N.		
3	ante	1	2 december	1898993	1.91010100	1	1			
2	-atras			1.00110-000			en a la	¢		
3	ine .	2		100716		and the second	- Rel		10.00	_
	194	1	1.000004	1000000	LACOUR.	and the second second	-	-	<	

B B G K K Figure 4: Calculating Term Frequency

+			Carline	194	Tana 1	(F) (
1	and a			a la maint surrow.		2
2	a secondary		1.0	a providence	1777 (A-1946)	
2	-		-	1.	LITERING C	
	-		1.0000000000000000000000000000000000000	A TAXABLE	Lacesteine	
3			a chainean	6 ischeries).	capacity in the	
				a larmalinas.	1 fuerers.	
2	-		1.000	a Landson True.	a tar (asso.)	
	-		A Christmann	a lamadera.	1 LAUTON.	
A	and and			a interest	Derivate.	
3	instead of			a presentation	1475MER	
2	and an	+		a commission	DATION OF A	
a				10.000	1 (01) (100)	
a	1000			a manufacture		
9	income.			6. 11 Photosopher		
A	1000			a literenaria		
P	lates .			a comunita		
A		·		a transport		
	194		1.1	d immediate	LIESTN.	4

Figure 5: Calculating IDF

5. Conclusion

To find out a bug, developers use not only the content of the bug report but also field knowledge applicable to the software project. We introduce a learning-to-rank approach that emulates the bug discovering process affianced by developers. The ranking model qualify useful relationships between a bug report and source code files by leveraging field knowledge, such as API description, the syntactic artifact of code, or issue tracking data. Experimental measure on six Java projects show that our approach can out the relevant files within the top find 10 recommendations for over 70 percentage of the bug reports in Eclipse Platform and Tomcat. Moreover, the proposed ranking model outperforms three recent state-of-the-art approaches. Feature valuation research employing greedy backward feature elimination demonstrate that all properties are useful. When coupled with runtime analysis, the feature valuation results can be utilized to select a subset of properties in order to achieve a target trade-off between system accuracy and runtime complexity.

Reference

- [1] Y. Brun and M. D. Ernst, "Finding latent code errors via machine learning over program executions," in Proc. 26th Int. Conf. Softw. Eng., Washington, DC, USA, 2004, pp. 480-490.
- [2] B. Dit, A. Holtzhauer, D. Poshyvanyk, and H. Kagdi, "A dataset from change history to support evaluation of software maintenance tasks," in Proc. 10th Working Conf. Mining Softw. Repositories, Piscataway, NJ, USA, 2013 pp. 131–134.
- [3] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," in Proc. 6th IEEE Int. Working Conf. Mining Softw. Repositories, Washington, DC, USA, 2009, pp. 71-80.
- [4] T. Joachims, "Training linear SVMs in linear time," in Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA, 2006 pp. 217–226.
- [5] S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying software changes: Clean or Buggy?" IEEE Trans. Softw. Eng., vol. 34, no. 2, pp. 181-196, Mar. 2008.
- [6] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: A comparative study of generic and composite text models," in Proc. 8th Working Conf. Mining Softw. Repositories, New York, NY, USA, 2011, pp. 43-52.
- [7] R. Saha, M. Lease, S. Khurshid, and D. Perry, "Improving bug localization using structured information retrieval," in Proc. IEEE/ ACM 28th Int. Conf. Autom. Softw. Eng., Nov. 2013, pp. 345-355.
- [8] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng., New York, NY, USA, 2014, pp. 689-699.

O:

Author Profile



Ms. S. Gomathi is currently working as Assistant Professor in Department of Computer Science and Engineering. Her area of interest is Neural Networks, Data Mining.



Ms. K. Swathini is currently pursuing Bachelor's degree in Computer Science and Engineering at Sri Krishna College of Technology, Coimbatore. Her area of interest includes Software Engineering and Data

Structures.



Ms. M. Suvetha is currently pursuing Bachelor's degree in Computer Science and Engineering at Sri Krishna College of Technology, Coimbatore. Her area of interest includes Software Engineering and Database Management System.



Ms. M. Deepa is currently pursuing Bachelor's degree in Computer Science and Engineering at Sri Krishna College of Technology, Coimbatore. Her area of interest includes Software Engineering and Database

Management System.

