2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

Concept-based Classification of Software Defect Reports

Sangameshwar Patil
TRDDC, Tata Consultancy Services, Pune, India and
Dept. of Computer Science and Engg., IIT Madras, India
Email: sangameshwar.patil@tcs.com

***Abstract*—Automatic identification of the defect type from the textual description of a software defect can significantly speed-up as well as improve the software defect management life-cycle. This has been recognized in the research community and multiple solutions based on supervised learning approach have been proposed in the recent literature. However, these approaches need significant amount of labeled training data for use in real-life projects.**

In this paper, we propose to use Explicit Semantic Analysis (ESA) to carry out concept-based classification of software defect reports. We compute the "semantic similarity" between the defect type labels and the defect report in a concept space spanned by Wikipedia articles and then, assign the defect type which has the highest similarity with the defect report. This approach helps us to circumvent the problem of dependence on labeled training data. Experimental results show that using concept-based classification is a promising approach for software defect classification to avoid the expensive process of creating labeled training data and yet get accuracy comparable to the traditional supervised learning approaches. To the best of our knowledge, this is the first use of Wikipedia and ESA for software defect classification problem.

***Keywords*-Software Defect Classification, Explicit Semantic Analysis, Mining Software Respositories, Text Data Mining**

I. Introduction

A lot of important information is captured as text data in the software development life-cycle (SDLC). Software defect management is a vital part of maintenance and evolution phases of the SDLC. During the testing phase as well as real-life usage of software, many defects regarding various aspects of software are reported. Classifying these defects using a suitable defect classification scheme (such as the Orthogonal Defect Classification (ODC) [1], IEEE standard 1044 [2], HP classification scheme [3] etc.) helps to streamline the defect management process and reap multiple benefits such as identifying patterns in the defect reports, faster root cause analysis and so on.

The textual description in a software defect (i.e., software bug) report is very important for understanding of the defect and its subsequent classification as per a given classification scheme. Automatic identification of the defect type from the textual defect description can significantly improve the defect analysis time and the overall defect management process. This has been recognized in the software repository mining research community and multiple solutions have been proposed over the past decade [4]–[7].

The standard data-driven approach such as supervised machine-learning for software defect type classification needs a significant amount of labeled training data to build a predictive model [5], [6]. This labeled dataset is typically created by humans with domain knowledge and expertise. This is clearly an effort-intensive as well as expensive activity. The research community is aware of this challenge and has recently proposed more innovative solutions such as use of active learning and semi-supervised learning for software defect classification [8]. These more advanced approaches aim to reduce amount of labeled training data required and in-turn minimize the human annotation effort required. Even though these approaches improve upon the basic supervised learning approach, they still need reasonable human effort to produce the necessary amount of labeled training data to carry out the software defect classification.

In this paper, we explore a different approach to avoid the labeled training data and still achieve the automated classification of software defects using the "semantic" information inherent in the label descriptions. We propose to use keywords from the defect labels' textual descriptions in the defect classification schemes (such as ODC) and represent the labels using Wikipedia[1] articles as features. Following Gabrilovich et al. [9], we assume that each Wikipedia article corresponds to a human-interpretable *concept* and we refer to the vector space spanned by all articles in Wikipedia as the *concept space*. Using Explicit Semantic Analysis [9], we project the defect labels as well as individual defect descriptions in this *concept space*. Thus, instead of the traditional bag-of-words features [10], we represent the classification labels as well as the individual defects using this *bag-of-concepts* feature representation.

We compute similarity between the defect labels and a defect description in the Wikipedia concept space and then, assign the defect label which has the highest similarity with that defect description. This approach helps us to circumvent the problem of dependence on labeled training data. Using the standard benchmark defect classification dataset [6], we show that the proposed approach achieves comparable weighted classification accuracy with current state-of-the-art software defect classification method [8], that uses active semi-supervised learning to minimize the amount of labeled training

[1]https://en.wikipedia.org

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.20

data.

Rest of the paper has been organized as follows: In Section II, we briefly review the related work. In Section III, we provide details of our proposed approach, followed by its experimental evaluation in Section IV. Finally, we conclude with pointers to future work in Section V.

II. Related Work

Automating different tasks in the defect management process using techniques from machine learning and natural language processing has been receiving increasing attention from the research community. Defect type classification is a vital sub-task of the overall bug triage process. Bug triage process aims to assign unresolved bug reports to appropriate developers. There is a rich body of work on bug triage using text mining and machine learning techniques over the past decade [4], [11].

Huang et al. [5], [7] developed the AutoODC text classification tool and used it for classifying software defects along the "impact" dimension of ODC classification scheme. They have used the well-known Support Vector Machine (SVM) algorithm and shown viability of supervised learning approach for software defect classification task on a proprietary dataset.

Thung et al. [6] created a public benchmark dataset for software defect classification and compared various classification algorithms. They focus on classifying defect reports using three high-level defect types: control-flow, structural and non-functional defects. The first two types viz. control-flow and structural defect types have been synthesized from the original seven defect types available in the Orthogonal Defect Classification scheme. The non-functional (also called as *non-code* in [8]) defect type is used to augment the defect types provided by ODC with the defects in configuration and documentation related to software. They compare performance of classifier algorithms such as C4.5, Logistic regression, Naive Bayes, RBF network and Support Vector Machine (SVM). They find that the SVM multi-class classifier gives the best results with F-1 measure of 0.692 when using 90% (i.e., 450 defects) of their labeled data from benchmark dataset for training and 10% (i.e., 50 defects) for testing purpose.

In their recent work, Thung et al. [8] tackle the challenge of minimizing human effort required for creating labeled training data and advocate use of more advanced machine learning techniques of active learning and semi-supervised learning. In this approach, instead of asking human to label a large fraction of data (for instance, 90% in [6]), the human effort is utilized to label only those defects which are most informative to build the predictive model.

In this paper, our goal is same as that of Thung et al. [8], i.e., to minimize the human effort required to automate the software defect classification process. It is worth noting that Thung et al. [6], [8] use two kinds of features for training the various supervised learning algorithms: (i) textual features from the content of the defect report and the defect labels and (ii) code features obtained by pre-processing the code that fixes the bug. In our approach, we use only the textual data of the software defect report and do not need access to any related source code.

III. Concept-based Classification of Software Bug Reports

Our aim is to reduce the need for labeled training data to carry out defect classification. As observed by Gabrilovich et al. [9], we note that when human experts carry out text classification, they do not necessarily look for labeled training data. Humans seem to use inherent semantic information available in the textual description of defect report as well as the textual label of the defect type. This idea has been termed as Explicit Semantic Analysis [9], [12]. ESA is based on the hypothesis that we (humans) judge the semantic relatedness of two words or two documents at a conceptual level. For instance, we treat *apple* and *orange* as more similar or more semantically related to each other when compared to *apple* and *movie*. While comparing the words (or defect type labels in our case) we are essentially working with a set of concepts associated with them using significant background knowledge available in form of Wikipedia. Semantic similarity of two words is high if they share a closely coupled set of concepts.

Similar to ESA, we propose to first compute the "semantic similarity" between the defect type labels and the defect report description in a concept space spanned by Wikipedia articles and then, assign the defect type which has the highest similarity with the defect report description in this concept space. This approach helps us to circumvent the problem of dependence on labeled training data.

A. Details of Our Approach

Notation: We use the following notation to describe the proposed approach:

- $D = \{w_d\}$ set of words in the input text data of a defect *report*.
- $L = \{w_l\}$ set of words in the input text data of a defect *label*.
- $\overrightarrow{v_i}$ = TF-IDF vector (for D or L, as the case may be), where v_i is the weight of word w_i.
- $\overrightarrow{k_j}$ = an inverted index entry for word w_i, where k_j quantifies the strength of association of word w_i with Wikipedia concept c_j;
- N = number of Wikipedia concepts used in ESA representation (ordered by decreasing strength of association)
- $p_j = \sum_{w_i \in D} v_i.k_j$ = weight of each concept c_j in ESA representation of D.
- $\overrightarrow{V_D} = [p_1, \ldots, p_N]^T$; ($\overrightarrow{V_D}$ is ESA representation vector of length N for D)
- $q_j = \sum_{w_i \in L} v_i.k_j$ = weight of each concept c_j in ESA representation of L.
- $\overrightarrow{V_L} = [q_1, \ldots, q_N]^T$; ($\overrightarrow{V_L}$ is ESA representation vector of length N for L)

Proposed Approach:

A defect report D is represented using a vector $\overrightarrow{V_D}$ of Wikipedia articles using ESA. Each defect type label is also represented using a corresponding ESA vector $\overrightarrow{V_L}$. The defect

report is assigned the defect type whose ESA representation is closest to the ESA representation of the defect report.

The ESA concept space is spanned by the set of articles in Wikipedia. A concept c_j in ESA representation corresponds to a Wikipedia article and is represented as a TF-IDF weighted vector of words in that article. An inverted index is prepared which helps in mapping a word to a list of concepts in which it appears. Word-to-concepts associations in this inverted index are weighted by TF-IDF. Relatively low weight entries in this inverted index are filtered as noise. Given a defect report D, it is first represented as a term-vector using bag of words scheme weighted by TF-IDF. Using the inverted index, we merge the concept vectors for each term $w_d \in D$ to form the weighted vector of concepts representing the given defect report.

Entries of $\overrightarrow{V_D}$ (similarly, $\overrightarrow{V_L}$) reflect the relevance of the corresponding concepts to the input defect report D (similarly, L). To compute semantic relatedness of a defect type label L and a defect report D, we compare their ESA representation vectors using the cosine metric.

$$sim_{ESA}(\overrightarrow{V_D}, \overrightarrow{V_L}) = \frac{\overrightarrow{V_D} \ . \ \overrightarrow{V_L}}{||\overrightarrow{V_D}|| \ . \ ||\overrightarrow{V_L}||}$$

For the final classification, the defect report is assigned the defect type whose ESA representation is closest to the ESA representation of the defect report.

B. An Illustrative Example

Consider the following sample bug report[2] (LUCENE-1805) from Apache JIRA repository for Lucene library[3]

(Summary/Title): `CloseableThreadLocal should allow null Objects`

```
  CloseableThreadLocal does not allow null
Objects in its get() method, but does nothing
to prevent them in set(Object). The comment in
get() before assert v != null is irrelevant - the
application might have passed null. Null is an
important value for Analyzers. Since tokenStreams
(a ThreadLocal private member in Analyzer) is not
accessible by extending classes, the only way for
an Analyzer to reset the tokenStreams is by calling
setPreviousTokenStream(null). I will post a patch w/
a test.
```

Table I shows the top-10 concepts (i.e., Wikipedia articles) identified by ESA as having highest similarity with description of LUCENE-1805 defect report. Similarly, we project the defect label descriptions from ODC scheme for the 3 defect types (*control and data flow, structual, and non-code*; as synthesized by Thung et al. [6]) and get their ESA representations as shown[4] in Table II. From Tables I and II, we observe that LUCENE-1805 defect shares more concepts with the *control and data flow* defect type and these common concepts have higher ESA weight (q_j) than other defect types. This explains the intuition behind assignment of *control and data flow* defect label to LUCENE-1805 defect by the proposed approach.

[2]https://issues.apache.org/jira/browse/LUCENE-1805

[3]https://issues.apache.org/jira/browse/LUCENE

[4]Note that, due to space constraints, only top-10 concepts ordered by the highest ESA weight are shown.

TABLE I
TOP-10 CONCEPTS (I.E., WIKIPEDIA ARTICLES) IDENTIFIED BY ESA AS HAVING HIGHEST SIMILARITY WITH THE LUCENE DEFECT #1805

Concept (Wikipedia Article)	**ESA Weight (p_j)**
Comparison of C Sharp and Java	0.1836
Java syntax	0.1691
Null Object pattern	0.1638
Null (SQL)	0.1453
C 11	0.1438
Objective C	0.1426
Scientific method	0.1345
Pointer (computer programming)	0.1316
Constructor (object oriented programming)	0.1176
C Sharp syntax	0.1109

IV. EXPERIMENTAL EVALUATION

Experimental Setup: As the knowledge base for ESA, we used the English Wikipedia dump from 14 April 2016. We indexed 3,334,509 Wikipedia documents which act as the dimensions of the concept-space used by ESA. We used top-50 wikipedia articles ordered by ESA weight (p_j for defect report D and q_j for defect type label L) to represent D and L in the concept space (i.e., $|\overrightarrow{V_D}| = |\overrightarrow{V_L}| = 50$). The indexing was carried out on a Xeon-Class server (3 GHz quad-core CPU, 32 GB RAM) which took about 30 hours to complete the indexing. The index needs about 30 GB of disk space and once ready, it needs to be available through out the classification process. Note that the knowledge-base indexing is a part of pre-processing step required to carry out concept-based classification and it needs to be done only once (assuming the same knowledge-base will be used in future). Once the pre-processing of the knowledge base was completed, approximately 109 minutes were required for projecting the defect labels and the 500 defect reports in the benchmark dataset and the actual classification step.

Dataset: We used ESA on the benchmark dataset and defect classification scheme created and used by Thung et al. in [6], [8]. Brief details of their defect classification scheme are presented in Section II (Related Work). Thung et al. [6] collected 500 defects from three popular software products: Mahout, Lucene, and OpenNLP. Defects from the JIRA repositories of the respective software systems were randomly selected: 200 from Mahout JIRA repository, 200 from Lucene JIRA repository, and 100 from OpenNLP JIRA repository.

Experimental Results: Table III compares the proposed approach with the state-of-the-art for the software defect type classification problem. Table IV shows the accuracy in terms of per-class as well as weighted-average precision, recall and F1 measure. As we can observe from Table IV, the *control and data flow* defect type achieves highest accuracy and the *structural* defect type has lowest accuracy. This accuracy pattern is similar to the state-of-the-art [6], [8].

TABLE II
TOP-10 WIKIPEDIA ARTICLES IDENTIFIED BY ESA AS HAVING HIGHEST SIMILARITY WITH THE SOFTWARE DEFECT TYPE LABELS. (THE ASTERISK (*) DENOTES A COMMON CONCEPT BETWEEN THE ESA REPRESENTATIONS OF A DEFECT LABEL AND THE LUCENE DEFECT #1805)

Control and data-flow Detect Type	
Concept (Wiki Article)	**ESA wt.** (q_j)
*Scope (computer science)**	*0.745*
Assignment (computer science)	0.6384
*C 11**	*0.6323*
Fortran 95 language features	0.5696
*C syntax**	*0.5607*
Environment variable	0.4547
*C Sharp syntax**	0.4484
*C**	0.4414
Parameter (computer programming)	0.4411
Lazy initialization	0.4192

Structural Defect Type	
Concept (Wiki Article)	**ESA wt.** (q_j)
Object Process Methodology	0.2662
Object oriented programming	0.2196
*Class (computer programming)**	*0.2032*
Function object	0.1988
Ceramic capacitor	0.1985
Microsoft Excel	0.193
Normalized Systems	0.1896
Factory (object oriented programming)	0.1856
*C 11**	0.1856
Organizational culture	0.1818

Non-code (Non-functional) Defect Type	
Concept (Wiki Article)	**ESA wt.** (q_j)
Coding conventions	0.1626
High Efficiency Video Coding	0.1238
Apache Maven	0.1124
Huffman coding	0.0995
Video Coding Experts Group	0.0971
Extreme programming practices	0.0883
MPEG 1	0.0849
Robert C Seacord	0.0804
Daala	0.0772
Adaptive Multi Rate Wideband	0.0747

TABLE III
COMPARISON WITH STATE-OF-THE-ART FOR SOFTWARE DEFECT CATEGORIZATION PROBLEM (ACCURACY MEASURES: WEIGHTED AVERAGES OF PRECISION (P), RECALL (R) AND F1)

Algorithm	**Wt.Avg P**	**Wt.Avg R**	**Wt.Avg F1**	**Features used**	**Amount of labeled training data**
Concept-based Classification (our approach)	0.4873	0.4897	0.4876	Only text data from label descriptions & the defect reports	Only label descriptions with key words from ODC definitions. No additional labeled training data required.
Supervised Learning Approach using SVM [6]	0.69	0.7	0.692	Text data & **source code** features	90% of examples from the dataset (10-fold cross validation)
LeDEx: Active and Semi-Supervised Learning [8]	0.651	0.669	0.623	Text data & **source code** features	50 labeled examples (i.e., 10% of the dataset) as Active Learning budget and 25 labeled examples (i.e., 5%) as initial sample from clustering step

TABLE IV
PER-CLASS ACCURACY OF CONCEPT-BASED CLASSIFICATION ON THE BENCHMARK DATASET (PRECISION (P), RECALL (R) AND F1-MEASURE)

Software Defect Type	**P**	**R**	**F1**	**Fraction of examples in the benchmark dataset**
Control and Data Flow	0.6087	0.6311	0.6197	286/500 = 0.572
Non-code (refered as *non-functional* in [6])	0.4146	0.3434	0.3757	104/500 = 0.208
Structural	0.2404	0.2604	0.25	110/500 = 0.22
Weighted Average	0.4873	0.4897	0.4876	

In Table III, observe that the SVM-based supervised learning approach [6] needs 90% of input data to be labeled and the more advanced approach of active learning and semi-supervised learning [8] needs 15% of input data to be labeled. Further, [6], [8] also need access to the source-code of the software, but the proposed method does not need access to the source-code of the software. We note that the proposed approach needs only the defect label descriptions, which are typically available from the defect classification scheme (e.g., ODC) being used and does not need any labeled training data. Hence, even though the accuracy of proposed method is slightly lower than the state-of-the-art, using concept-based classification is a promising approach for software defect classification.

V. CONCLUSION AND FUTURE WORK

Quick and accurate classification of software defect reports is important for software defect management process of an enterprise or an open-source project. Current state-of-the-art approaches for automated software defect classification use supervised, semi-supervised and active learning approaches which need expensive labeled training data creation and hence often have limited use in practice. In this paper, we explored a novel approach to avoid the labeled training data creation step and still achieve the automated classification of software defects. We proposed use of Explicit Semantic Analysis to carry out concept-based classification of software bug reports. We compute the "semantic similarity" between the defect type labels and the defect report in a concept space spanned by Wikipedia articles and then, assign the defect type label which has the highest similarity with the defect report. This approach helps us to circumvent the problem of dependence on labeled training data. To the best of our knowledge, this is the first use of Wikipedia and ESA for software defect classification problem. Experimental results show that using concept-based classification is a promising approach for software defect classification.

As part of future work, effect of number of concepts (i.e., parameter N) used in ESA representation on the classification accuracy needs to be investigated. Tuning the value of N to improve the accuracy of concept-based software defect classification is an important direction for future work.

REFERENCES

[1] R. Chillarege, "Orthogonal defect classification," *Handbook of Software Reliability Engineering*, pp. 359–399, 1996.

[2] IEEE, *IEEE Standard 1044-2009 Classification for Software Anomalies*, 2009.

[3] S. Wagner, "Defect classification and defect types revisited," in *Proceedings of the 2008 workshop on Defects in large software systems*. ACM, 2008, pp. 39–40.

[4] D. Čubranić, "Automatic bug triage using text categorization," in *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer, 2004.

[5] L. Huang, V. Ng, I. Persing, R. Geng, X. Bai, and J. Tian, "Autoodc: Automated generation of orthogonal defect classifications," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2011.

[6] F. Thung, D. Lo, and L. Jiang, "Automatic defect categorization," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, 2012, pp. 205–214.

[7] L. Huang, V. Ng, I. Persing, M. Chen, Z. Li, R. Geng, and J. Tian, "Autoodc: Automated generation of orthogonal defect classifications," *Automated Software Engineering journal*, vol. 22, no. 1, pp. 3–46, 2015.

[8] F. Thung, X.-B. Le D., and D. Lo, "Active semi-supervised defect categorization," in *IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 60–70.

[9] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis." in *IJCAI*, vol. 7, 2007, pp. 1606–1611.

[10] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008.

[11] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *Journal of Software*, vol. 8, no. 9, pp. 2185–2190, 2013.

[12] O. Egozi, S. Markovitch, and E. Gabrilovich, "Concept-based information retrieval using explicit semantic analysis," *ACM Transactions on Information Systems (TOIS)*, vol. 29, no. 2, p. 8, 2011.