

Autoclassify Software Defects Using Orthogonal Defect Classification

Sushil Kumar[1(✉)], Meera Sharma[2], S. K. Muttoo[3], and V. B. Singh[4]

[1] Shyam Lal College, University of Delhi, Delhi, India
kumar.sk106@gmail.com
[2] Swami Shraddhanand College, University of Delhi, Delhi, India
meerasharma@ss.du.ac.in
[3] Department of Computer Science, University of Delhi, Delhi, India
[4] School of computer and systems sciences, JNU, New Delhi, India
vbsingh@mail.jnu.ac.in

Abstract. Software systems have become an integral part of all the organizations. These systems are performing many critical operations. A defect in these systems affects the product quality and the software development process. Prediction of the impact category of these defects helps in improving defect management process as well as taking correct decisions to fix defects. Orthogonal defect classification is a popular model for classifying defects and it provides an in-depth analysis of the defects. In this study, we proposed an auto classify approach to classify the defects into impact categories as defined by Orthogonal Defect Classification (ODC). Bag of words, term frequency-inverse document frequency and word embedding have been used to represent the textual data into numeric vectors. For experimental work, we have used 4,096 reports form three NoSQL databases. We have trained and tested the proposed autoclassify approach using Support Vector Machine (SVM) and Random Forest Classifier (RFC). We achieved maximum accuracy 94% and 85.99% using SVM and RFC respectively.

Keywords: Orthogonal defect classification · Bag of words · Term frequency inverse document frequency · Word embedding

1 Introduction

Business organizations are using software to record and maintain data about their employees, products and various operations such as financial and database management. Predicting the impact of these defects can help us to save cost and time in software development as well as in improving software defect prediction and classification models. ODC [1, 2] is a well-defined framework for analysing and classifying defects during software development. It is a systematic approach that provides a valuable feedback method for assessing the different phases of the software development. ODC can be applied in any phase to capture the semantic of each defect quickly. It characterizes the defects into eight attributes by knowing cause and its effect on the quality of software product. The ODC attributes include defect type, impact, trigger, activity, target, source,

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13381, pp. 313–322, 2022.
https://doi.org/10.1007/978-3-031-10548-7_23

qualifier and age. Defect type denotes change made to solve or correct a problem. The change can be in a function, condition and assignment. Impact refers to an experience of a user when the defect was surfaced. Impact can be capability, reliability, usability etc. Activity refers to the action performed when a defect was disclosed. Code inspection and function test are the actions fall under the activity attribute. Target refers to the object of the correction i.e., the build scripts or package. Trigger is a condition which makes a defect to occur. Qualifier describes the code point before a correction was made. Source refers to the origin of the defect i.e., the location of the defect where it was initially occurred. Age refers to the occurrence of a defect to the correction of another defect.

ODC is a systematic approach which helps in prioritizing the defects and assigning defects to the experts. It also enables managers to take correct decisions after they know the category of the defect. It helps in improving software defect prediction techniques. The systematic classification of software failures allows us to know the impacts on the software product quality. ODC further divides the impact attribute into 13 categories to understand the nature of impact. Impact category includes usability, performance, capability, installability, documentation, capacity, migration, serviceability, standards, integrity/security, serviceability, and reliability.

Bridge and Miller [3] presented a methodology that successfully applied the ODC to classify the defects found in Motorola GSM division. They successfully applied the orthogonal defect classification to software development group. Zheng et al. [4] applied ODC to analyse the software defects by adopting defect type attributes with the development process.

The limitation of the ODC is the manual classification of defect reports which is a time taking process and involves high cost. This process of manual classification becomes a complex task when the number of defect reports increases. Hence, there is a need to automate this process. Therefore, we propose an autoclassify classification approach to automate the process of defect classification based on ODC impact attributes.

The important highlights of the proposed autoclassify approach are:

- Defect reports classification using textual description of the defect;
- Model the classification approach into a multiclass text classification;
- Bag of words, term frequency inverse document frequency and GloVe word embedding were employed;
- 4,096 defect reports from Hbase, MongoDb and cassandra;
- Synthetic Minority Over Sampling Technique (SMOTE) for balancing the data distribution among ODC impact categories;
- The proposed approach evaluate the performance of SVM and RFC;
- SVM and RFC classify the defects with accuracy of 94% and 85.99% respectively.

The remainder of the paper is divided into six parts. ODC based related work has been introduced in Sect. 2. Proposed approach and results are discussed in Sects. 3 and 4 respectively. Section 5 presents the threats to validity of the work and finally work is concluded in the last section.

2 Related Work

ODC defect type and impact have been widely used in literature to classify the defect reports. Some proposed studies have used a single attribute such as impact or type while others have focused on multiple ODC attributes. A common problem is that the defects are manually classified by the experts or researchers in the field which is very tedious and time consuming process. Hence, there is a need to automate the process of classification. A classification approach has been introduced in literature, which classifies defects automatically by grouping ODC defect types in categories like non-functional, control and data flow and structural [5]. The categories Algorithm/Method, Assignment/Initialization, Checking, Timing/Serialization are grouped into control and flow. Function/Class/Object, Interface/O-O Messages and Relationship are grouped into Structural. Remaining defect types are grouped as non-functional. Total 500 defect reports were collected and labeled manually. The proposed approach achieves an average accuracy of 77.8% using support vector machine. The authors used the expert's experience to label the defects and evaluate the performance on minimal labeled data.

In another study [6] an active semi supervised approach has been proposed to minimize the effort of manual labeling. Out of 500 defect reports, only 50 defect reports were manually labeled. The proposed approach achieved a weighted precision, recall, f-measure and area under curve (AUC) score of 65.1%, 66.9%, 62.3% and 71% respectively.

In [7], authors suggested a method for automatically classifying software bugs based on attributes retrieved from the Abstract Syntax Tree (AST) of the source code. The approach was assessed by labeling the 1,174 defect reports into four categories, Computational, Control or Logic, Data and Interface. The experimental results show the highest accuracy, precision and recall of 79%, 75% and 73% respectively. With minimum number of classes, the classification task seems to be easy as compared to the eight ODC attributes and their sub-attributes.

An Auto ODC technique to classify the software defects according to ODC defect impact attribute has been presented in [8]. The technique combines ODC expertise with domain knowledge. The authors also proposed an annotation framework to assign a category with justification.

In [9], the authors presented a software defect classification approach using crowd labeling for 1,444 defect reports from two datasets compendium and Mozilla, and labeled according to ODC defect impact by set of 5 annotators having no expertise in the field. The authors used 4 out of 14 impact categories to train the classifier.

Recently, in [10] authors presented an automatic classification of software defects using supervised machine learning algorithms. For experimental work, Authors collect and manually labels the 4,096 defect reports according to all ODC attributes. As opposed to the previous work, we have generated a new word embedding along with pre trained embedding GloVe by Stanford. Similar work [11] discusses the effectiveness of word embedding techniques to envisage the intensity of defect in the software product. Similarly authors [12] presented a bug severity assessment to predict the forecity of the bug reports in the context of cross project. The previous research has employed multiple ODC defect attributes to automate the process of defect classification. Moreover, the exiting works use the Tf-Idf or Bag of Words (BoW) model to represent the words in

numerical vectors. In this work, we generate numerical vectors for defect corpus using GloVe word embedding [13]. Word embedding represents the similar words into similar vector representation and hence able to capture the semantic of the words. Recently word embedding techniques have been used in [14–17].

3 Proposed Approach

The proposed approach can be viewed as a 5 step process from collecting dataset to get the results shown in Fig. 1.

Step 1: Collecting the dataset
We have collected a dataset consisting of 4,096 defect reports from http://odc.dei.uc.pt/. The dataset contains software defect reports from Hbase, Cassandra and MongoDB. The defects are labeled by the researchers by considering the six orthogonal defect attributes, namely defect type, Trigger, Activity, Impact, Target and Qualifier. For this study, we use the impact attribute that is further divided into following categories- Capability, Usability, Installability, Maintenance, Integrity/Security, Documentation, Serviceability, Performance, Reliability and Requirements.

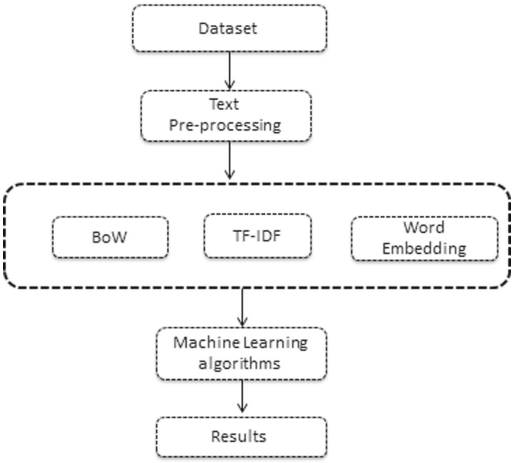

Fig. 1. Overview of the proposed approach

Step 2: Text Preprocessing
The dataset contains two fields summary and impact categories. Each defect report contains a textual description about the defect. For example, a defect report with summary "*misspelled columsort attribute* results *deleted commit logs db files*" has been labeled with "*Reliability*" category. We preprocess the text by applying text mining techniques such as removing punctuation and stop words, converting into lowercase and tokenize the text. Stop words were removed using stop word provided in NLTK library. Finally, we get the cleaned and tokenized text.

Step 3: Text to numbers
We have used three schemes- Tf-Idf, GloVe word embedding and BoW, to construct the numeric vector from tokenized corpus. In bag of words, a binary vector is created for each word in each report. If a word is present in the report, it is assigned 1 otherwise 0. Bag of words is a count based method.

Tf-Idf is frequency based method which specifies the importance of a word in a report. Tf refers to the term frequency and calculated using Eq. (1). The inverse document frequency is calculated using Eq. (2). Finally, Tf-Idf is the product of Tf and Idf score as shown in (3).

$$Tf_{i,j} = \frac{\#times \;\; a \;\; word \;\; i \;\; appear \;\; in \; a \;\; document \;\; j}{\#total \;\; document} \tag{1}$$

$$Idf_i = log \frac{\#total \;\; document}{\#document \;\; containingi} \tag{2}$$

$$TfIdf \; = Tf \times Idf \tag{3}$$

Word embedding is a process to represent the text into word vectors of fixed size. The idea behind word embedding is that words having similar meaning have similar representation. We use GloVe word embedding in this work. GloVe word embedding model [13] is based on the word occurrences. We downloaded the GloVe pre trained word2vec model from http://nlp.stanford.edu/data/glove.6B.zip. It contains 1.2 million word embedding. Pre trained GloVe word embedding is used in many classification tasks [18, 19]. It has been used recently for text classification [25–28].

Step 4: Machine learning algorithms
In this work, we have used two classifiers SVM and RFC. SVM [20, 21] is most widely used supervised classification algorithm. SVM is a binary classifier that separates two classes by finding the optimal hyper plane by minimizing the cost function between the feature points using Eq. (4) and (5).

$$min \frac{1}{2} \left\| w^2 \right\| + C \sum \xi_i \tag{4}$$

$$\text{s.t } y_i \left(w^T x_i + b \right) \geq 1 - \xi_i \tag{5}$$

where C is the parameter between margin and error. ξ_i represents the measure of training. But it can be extended to a multiclass classification problem by using the combination of several binary support vector machine classifiers. Many methods exist for modeling a multiclass problem into a series of n binary sub problems. For this study, we have used one against all approach (OAA) in which the ith classifier is trained to differentiate the examples belong to i^{th} class from rest of the examples. Because of the variety of kernel functions utilized, SVM can handle the high volume data. SVM has been utilized in variety of investigations.

A set of random trees is used in the Random Forest classifier (RFC) [22]. It selects the features at random to create distinct trees. The bagging method [23] is proposed in which, distinct trees are trained parallelly depending on the supplied information.

Bootstrapping aggregation guarantees that each tree in the random forest is distinct lowering the classifier's variance. RFC can accurately classify massive amounts of data. Because of its random character, it is less prone to over fitting. In comparison to other algorithms, RFC yields more consistent results.

Step 5: Results
The classification performance is presented using four evaluation metrics, namely accuracy, precision, recall and F1 score.

4 Experimental Setup

This section presents the dataset details and performance evaluation.

4.1 Dataset

For the experimental work, we used the same dataset as used in [10]. The dataset is publically available and it is verified by the two researchers. The agreement between these researchers is calculated using kohen kappa agreement. For the impact ODC attribute, the kohen kappa value is 82% which is almost perfect. The dataset details are outlined in Table 1. The impact attribute is further divided into 13 categories. In this study, we have used only 6 categories- capability, integrity/security, performance, reliability, requirements and serviceability. The rest of the categories have assigned to very few reports.

Table 1. Dataset details

Defect impact	#Reports
Capability	2,165
Integrity/Security	87
Performance	199
Reliability	988
Requirements	227
Serviceability	156

Comments, title and description are the part of the defect report which helps in defect report classification manually into orthogonal defect classification impact attributes. Researchers have assigned the capability category to 50% of the data. The defect impact categories are varying proportionally and lead to an imbalance problem. To tackle this, we use Synthetic Minority Over Sampling Technique (SMOTE) [24] for balancing the data distribution for the defect impact categories.

4.2 Evaluation Metric

We evaluated the efficiency of our proposed approach using four metrics like F1 score, precision, accuracy and recall. A confusion matrix consisting of actual and predicted values is used to calculate these four metrics: accuracy, Precision, Recall and F1 score are calculated as follows;

$$\text{Accuracy} = \frac{tp_c + tn_c}{tp_c + fp_c + tn_c + fn_c} \tag{6}$$

$$Precision = \frac{tp_c}{tp_c + fp_c} \tag{7}$$

$$Recall = \frac{tp_c}{tp_c + fn_c} \tag{8}$$

$$F1 = 2 * \frac{Precision_i * recall_i}{Prcision_i * recall_i} \tag{9}$$

where tp_c and tn_c represents the true positive and true negative which is the number of instances correctly and not correctly respectively classified as class c; while they belong to some other classes. fp_c represents the false positive which is the number of instances correctly not classified as c; tp_c represents the true negative which is the number of instances assigned to some other classes while they belong to class c.

5 Results

We evaluated the performance of two classifiers SVM and RFC by learning features using three methods BoW, Tf-Idf and GloVe word embedding. The results obtained by these classifiers in terms of accuracy, precision, recall and F1 are presented in Table 2. In Bag of words method, each defect report is represented as numerical vector consists of total count i.e. the number of times a word present in a particular defect report. A total of 23,530 unique words are present in the defect corpus and forms a feature matrix where row represent the defect reports while columns are the unique words. The proposed approach achieves accuracy of 64.03% for RFC and 62.15% for SVM. Tf-Idf is a frequency based method to convert text into numeric vectors. The Tf-Idf score is inversely proportional to the occurrence of a word in the document. Our approach achieves an accuracy of 85.99% and 94% for RFC and SVM. We have used GloVe pre trained word embedding to get the real valued vectors of fixed dimension. The dimension for each vector is taken as 300. We downloaded the GloVe word embedding from http://nlp.stanford.edu/data/glove.6B.zip. We fed our defect corpus to GloVe to generate the numerical vectors for the words present in the corpus. Using these vectors, RFC and SVM are able to achieve 72.26% and 81.43% accuracy. There are some tunable parameters associated with classifiers to get the optimum results. GridSearchCV is used to search the best parameter for both the classifiers- RFC and SVM. For RFC, we search for the number of trees. The number of trees for all the experiments varies from 128 to 1,024. For SVM, we search for C and for C = 1 we get the highest accuracy, precision, recall and F1 score. We carried out all the experimental work in Python.

Table 2. Accuracy, precision recall and F1 (%)

Method	Classifier	Accuracy	Precision	Recall	F1
BoW	RFC	64.03	60.28	58.81	54.61
	SVM	62.15	59.64	54.09	52.82
TF-IDF	RFC	85.99	86.21	85.20	82.68
	SVM	94	95.08	93.99	93.85
GloVe	RFC	72.26	71	62.24	60.11
	SVM	81.43	79.67	74.56	71.38

6 Threats to Validity

In terms of external threats, an obvious threat is the training from word vectors which is generated for the thousands of defect reports. But the word vectors can be further improved. The distribution of the classes in the datasets is not uniform. The chance of biased learning is high. We attempted to eliminate the bias. The external threat is also concerned about the size of defect reports and brief description of each defect is also a concern to the external threat. In the future, this hazard can be mitigated in the future by using a larger dataset to provide more accurate results. In terms of construct validity, the issues in defect repositories such as defect duplication must be addressed. Only description of defect is extracted for all the three datasets Cassandra, Hbase and MongoDB. Moreover the dataset is transformed into 6-class classification problem due to the assignment of remaining classes to fewer defect reports. However more techniques should be adopted, to deal with multiclass imbalance problem. The internal validity is related with how the examined systems are represented and how well the classifiers function on the combined datasets. However, they differ for every combination of the datasets. That may be due to the discriminant ability of textual description of the defect. It can also be affected by the word embedding produced by pre trained word embedding model. As a result, machine learning algorithms must react to different contexts.

7 Conclusion

We present an autoclassify approach to automate the defect classification into impact categories as defined by ODC. For evaluating the efficiency of the proposed approach, we test the performance of support vector machine and random forest in terms of accuracy, precision, recall and F1 score on 4,096 defect reports collected from Hbase, MongoDb and Cassandra. We model the autoclassify approach as multiclass text classification problem. We represent the text into numeric vectors using pre trained word embedding GloVe, Term Frequency Inverse Document Frequency and Bag of Words. Word embedding is generated for the defect corpus using GloVe. SVM and RFC achieves highest accuracy 94% and 85.99% while learning form the frequency based method i.e. Tf-Idf. However the proposed approach can be further improved on large number of defect reports. In future, we will also evaluate the performance of proposed approach on other machine learning classifiers including Naïve Bayes and XGboost.

References

1. Chillarege, R., et al.: Orthongonal defect classification-a concept for in-process measurements. IEEE Trans. Software Eng. **18**(11), 943–956 (1992)
2. Chillarege, R.: Orthogonal defect classification. In: Lyu, M.R. (ed.) Handbook of Software Reliability Engineering, pp. 359–399. IEEE CS Press, Los Alamitos (1996)
3. Bridge, N., Miller, C.: Orthogonal defect classification using defect data to improve software development. Softw. Qual. **3**(1), 1–8 (1998)
4. Zheng, J., Williams, L.: Nagappan, N., Snipes, W., Hudepohl, J. P., Vouk, M. A.: On the value of static analysis for fault detection in software. IEEE Trans. Softw. Eng. ***32***(4), 240–253 (2006)
5. Thung, F., Lo, D. Jiang, L.: Automatic defect categorization. In 19th Working Conference on Reverse Engineering, pp. 205–214. IEEE (2012)
6. Thung, F., Le, X.B.D., Lo, D.: Active semi-supervised defect categorization. In: IEEE 23rd International Conference on Program Comprehension, pp. 60–70. IEEE (2015)
7. Liu, C., Zhao, Y., Yang, Y., Lu, H., Zhou, Y., Xu, B.: An AST-based approach to classifying defects. In: IEEE International Conference on Software Quality, Reliability and Security-Companion, pp. 14–21. IEEE (2015)
8. Huang, L., et al.: AutoODC: automated generation of orthogonal defect classifications. Autom. Softw. Eng. **22**(1), 3–46 (2015)
9. Hernández-González, J., Rodriguez, D., Inza, I., Harrison, R., Lozano, J.A.: Learning to classify software defects from crowds: a novel approach. Appl. Soft Comput. **62**, 579–591 (2018)
10. Lopes, F., Agnelo, J., Teixeira, C.A., Laranjeiro, N., Bernardino, J.: Automating orthogonal defect classification using machine learning algorithms. Futur. Gener. Comput. Syst. **102**, 932–947 (2020)
11. Kumar, L., Kumar, M., Murthy, L.B., Misra, S., Kocher, V. Padmanabhuni, S.: An empirical study on application of word embedding techniques for prediction of software defect severity level. In:16th Conference on Computer Science and Intelligence Systems (FedCSIS), pp. 477–484. IEEE (2021)
12. Singh, V.B., Misra, S., Sharma, M.: Defect severity assessment in cross project context and identifying training candidates. J. Inf. Knowl. Manag. **16**(01), 1750005 (2002)
13. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. EMNLP **14**, 1532–1543 (2014)
14. Sangwan, N., Bhatnagar, V.: Optimized text classification using deep learning. In: Goar, V., Kuri, M., Kumar, R., Senjyu, T. (eds.) Advances in Information Communication Technology and Computing. LNNS, vol. 135, pp. 293–302. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-5421-6_30
15. Sivakumar, S., Rajalakshmi, R.: Analysis of sentiment on movie reviews using word embedding self-attentive LSTM. Int. J. Amb. Comput. Intell. **12**(2), 33–52 (2021)
16. Bayer, M., Kaufhold, M., Reuter, C: A survey on data augmentation for text classification. arXiv preprint arXiv:2107.03158 (2021)
17. Rahimi, Z., Homayounpour, M.M.: TensSent: a tensor based sentimental word embedding method. Appl. Intell. **51**(8), 6056–6071 (2021). https://doi.org/10.1007/s10489-020-02163-8
18. Lu, X., et al.: MKPM: Multi keyword-pair matching for natural language sentences. Appl. Intell. **52**(2), 1878–1892 (2021). https://doi.org/10.1007/s10489-021-02306-5
19. Yue, C., Cao, H., Xu, G., Dong, Y.: Collaborative attention neural network for multi-domain sentiment classification. Appl. Intell. **51**(6), 3174–3188 (2020). https://doi.org/10.1007/s10489-020-02021-7
20. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)

21. Duan, K.B., Keerthi, S.S.: Which is the best multiclass SVM method? An empirical study. In: Oza, N.C., Polikar, R., Kittler, J., Roli, F. (eds.) MCS 2005. LNCS, vol. 3541, pp. 278–285. Springer, Heidelberg (2005). https://doi.org/10.1007/11494683_28
22. Breiman, L.: Random Forests. Mach Learn. **45**, 5–32 (2001)
23. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
24. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
25. Asudani, D. S., Nagwani, N.K., Singh, P.: Exploring the effectiveness of word embedding based deep learning model for improving email classification. Data Technol. Appl. **56**(1) (2022). https://doi.org/10.1108/dta-07-2021-0191. ISSN: 2514:9288
26. Hossain, T., Mauni, H.Z., Rab, R.: Reducing the effect of imbalance in text classification using SVD and GloVe with ensemble and deep learning. Comput. Inform. **41**(1), 98–115 (2022)
27. Ebrahimi, F., Tushev, M., Mahmoud, A.: Classifying mobile applications using word embeddings. ACM Trans. Softw. Eng. Methodol. (TOSEM) **31**(2), 1–30 (2021)
28. Kirelli, Y., Özdemir, Ş. Sentiment classification performance analysis based on glove word embedding. Sakarya Univ. J. Sci. **25**(3), 639–646 (2021)