International Journal of Scientific Research in Science and Technology
Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)
doi : https://doi.org/10.32628/IJSRST523102143

Evaluating Deep Learning Methods for Classifying Bugs

Aryan Singh, Deepanshu Singhaniya, Dr. Ruchika Malhotra
Department of Software Engineering, Delhi Technological University New Delhi, India

ARTICLEINFO

Article History:

Accepted: 10 April 2023
Published: 06 May 2023

Publication Issue
Volume 10, Issue 3
May-June-2023

Page Number
29-35

ABSTRACT

Software maintenance is a crucial part of software development, especially now more than ever. Without proper maintenance, software can become outdated and unreliable. and vulnerable to security threats, which can have serious consequences for users and organisations that rely on it.

But as the software projects become larger, it becomes It is the responsibility of the managers to assign the bugs to the developers so that the developers can use their time efficiently in resolving those bugs. But the capacity of Managers' ability to analyse each and every bug report

and assign it to the appropriate developer is being out- paced by the shear number of bug reports, leading to slow progress. Its impossible for developers or managers to be able to understand hundreds of reports a week, let alone being able to have a good idea of each and every developer in the team to be able to appropriately assign the bugs to them.

Keywords: Software maintenance, Software development, Outdated software, Unreliable software, Security threats ,Bug reports, Bug resolution, Managerial responsibility, Developer efficiency, Bug assignment, Managerial capacity, Progress pace, Developer understanding, Bug management, Team management

I. INTRODUCTION

Software developers on large software projects often have a huge number of ever-increasing pending bug requests to fix. To ensure that the time of those developers is used effi- ciently, This paper proposes an automated approach to clas- sifying bugs. reports to the developers based on the previous bug fixes of each developer.aˇ A significant amount of an or- ganisations Time and money can be saved by automating the bug assignment process for its software projects.

In this project, we try to improve upon the existing ap- proach [1] of bug triaging using the DBRNN (deep bidi- rectional neural networks) approach by: (1) using GRU in place of LSTM in conjunction with RNN, (2) added more densely linked layers to effectively push the models performance ceiling. We discussed various approaches (in-cluding traditional machine learning and non-machine learning

approaches) and discussed a better and more accurate so- lution by developing a deep learning model. We explored a better rule-based method that can assist our deep learning model to get better accuracy [2].

II. Background

In software maintenance, bug assignment involves the as- signment of a defect or issue to a specific developer or team for analysis and resolution. When a bug is reported, it is entered into a bug tracking system, and a developer or team is assigned to investigate the issue. The assignment process considers factors such as the developers technical expertise, workload, and availability. Automating this task can vastly increase the development speed of a software project. This task can be automated in a variety of ways; the ones we ex- amined are as follows:

Figure 1. Bug Assignment overview

III.Brief

This paper is divided into three sections. The Introduc- tion section describes the traditional ways of assigning that our approach seeks to automate. It also describes the ways in which there may be significant cost savings due to faster processing and less human labour. The next two sections describe and compare existing methods and approaches to the various methods proposed by us. These methods are as follows:

1. Fuzzy Logic model
2. using Traditional Machine learning approaches
3. Tossing-graph model
4. Social-networks model
5. Deep learning using RNN with LSTM

Every method has advantages and disadvantages; the best approach would be to use any combination of best- performing models or ensemble learning models. Most of the relevant papers reviewed were ensemble learning ap- proaches.

IV. Literature Review

This section describes the various approaches recently used and proposed by the above reviewed work

Fuzzy Logic model

The method proposed in the works [2], [3] makes thorough use of fuzzy sets.“ Fuzzy implies a mathematical framework for dealing with uncertainty and imprecision. It is a type of logic that allows for reasoning in situations where the truth values of propositions are not clearly defined or when there is incomplete information. As the name suggests, the characteristics of a bug report are stored in the form of a ”fuzzy set.“ In order to extract the relationship between the bug report and the developer to be assigned to resolve it, A membership score is measured, which is then used to sort the bug reports and the developer profile. Each bug is then assigned to the appropriate developer with a matching mem- bership score. The fuzzy subset and rules are then adjusted to better optimise the systems speed and accuracy.

Fuzzy logic helps extract the ambiguous human logic from a body of text with the help of the member ship score without using any kind of machine learning, this also makes it much faster than any machine learning model.

The advantages of using fuzzy logic includes:

1. Simple
2. Light on resources
3. Faster
4. Easier to use and develop
5. Easier to get an understanding of the working of the model, unlike while using neural networks.

Traditional machine learning models(NB, SVM etc.)

The method proposed in the work [4] describes well the ways in which we can automate the bug classification pro- cess with little to no sentiment analysis. They mostly only used the Naive Bayes Model in conjunction with SVM to find correlations between the bugs and the developers who were assigned those bugs. Despite using SVM and naive bayes, the highest accuracy achieved was around 30% with average precision and recall values of 28%, which was far from ideal.

[5] developed a semi-supervised method of analysing the text and using the information obtained to further improve the ef- ficiency of supervised learning methods. They made use of the expectation maximisation (EM) algorithm, where there are two steps: In the E-step, the algorithm estimates the values of the missing or unobserved variables by ffect te- ing their expected values based on the available data and the current estimate of the model parameters. These expected values are also known as the "responsibilities" of each data point. In the M-step, the algorithm updates the estimates of the model parameters based on the calculated responsibilities from the E-step. The new parameter estimates are obtained by maximising the likelihood of the observed data given the estimated responsibilities. This repeats until the change in the parameter estimates between iterations falls below a cer- tain threshold; this further increased the accuracy to 36%, which was still not so powerful.

Another approach that got our attention was [6], where the researchers used multi-label KNN to find out about old bugs related to the bug reports being analysed. Then this informa- tion is used to determine the similarity between developers, and finally the KNN analysis and the developer similarity are combined to appropriately assign the bug in question. This method was able to achieve a significantly higher accuracy of 89.3%. The paper also compared other traditional machine learning approaches despite their poor efficiency, which in- clude SVM, the Naive Bayes model, and the C4.5 model, of which the SVM model performed the best with an accuracy of 64% on the firefox dataset.

Tossing-Graph models

The Tossing Graph Model (TGM) is a graph-based model used for defect assignment in software maintenance. It is based on the concept of "tossing," which refers to the process of passing a defect from one developer to another until it is resolved. The TGM represents the entire process of defect resolution as a graph, where each node represents a devel- oper and each edge represents the tossing of a defect from one developer to another. The TGM is used to predict the shortest path between the initial defect reporter and the final defect resolver, which helps speed up the process of defect resolution.

The TGM has been shown to be effec"ive 'n improv- ing the accuracy of automatic defect assignment compared to other machine learning-based models. It involves a se- ries of tossing steps, where a defect is passed from one developer to another until it reaches the final fixer. The model predicts a shorter path by identifying the most effi- cient path between the initial assignee and the final fixer. [7] showed that the tossing graph models outperform the tradi- tional machine learning models (particularly naive Bayes and Bayesian models) in some or most ways. In the paper [8], the naive Bayes model is combined with the tossing graph model to automate the target assignment, and the accuracy they achieved was far better than solely using the naive Bayes models, i.e., 86%. Overall, the tossing model is a promising approach to automated defect assignment, as it can improve the efficiency of the defect repair process and ultimately lead to better software quality.

Team Network model

As more and more people are collaborating digitally, it has become far easier to spot people with similar talents and skills. When people comment in chat

about a bug, the dis- cussions often hint at some correlation between other bugs or the developers fixing those other bugs. This information can be helpful in identifying the correlation between bugs and the developers of the project. In one such paper [9], the authors used a KNN model along with social network in- dicators to recommend a suitable defect repairer. They first search for historical defect reports that are similar to the new- found bugs using the KNN model. Then, they consider the reviewers involved in those historical defect reports as can- didates for repairing the newfound bugs. They compare the frequency and distribution of these indicators and find that the frequency and out-degree indicators can achieve the best results. In order to automatically assign defects, they used the naive Bayes model and SVM to get developer priority information and then analyse the bug reports to make a se- lection.

Other Custom models

[10] developed an approach using the source code vocab- ulary of the bug reports and then triaged the word vectors with that information to classify the bug reports. In the pa- per [11], a tool for automating the bug triaging process was developed that used historical information about the changes in the source code of the project and used that information to assign the detected defects. Using this approach, they were able to achieve a significant accuracy of 81.44%. There are quite a few ways in which (custom) expert models are superior to traditional machine learning or even deep learn- ing methodologies, primarily where efficiency is a concern. Expert systems are computer programmes that mimic the decision-making ability of a human expert in a specific do- main. These systems rely on expert knowledge and reasoning to provide advice or make decisions in a particular domain. There are several models of expert systems, each with its own approach to representing and using expert knowledge. The papers we described used the following models with differ- ing efficiencies and accuracies:

1. Rule-based expert systems
2. Fuzzy expert systems
3. Bayesian expert systems
4. Neural network expert systems
5. Case-based reasoning expert systems

Novel Deep Learning Approach(using RNN and GRU)

Traditional algorithms are rule-based approaches that rely on well-defined mathematical equations, heuristics, or logic to perform tasks. These algorithms require a lot of domain knowledge and expert feature engineering to work ffect- tively. They are often interpretable and can provide insights into how decisions are made. Examples of traditional algo- rithms include decision trees, support vector machines, and linear regression.

On the other hand, deep learning is a subfield of machine learning that uses neural networks to learn features and make decisions. These networks are composed of multiple layers of interconnected nodes that can identify complex patterns in the data without the need for explicit feature engineering. Deep learning models are often black boxes, meaning it can be challenging to understand how they arrive at their deci- sions. Examples of deep learning models include convolu- tional neural networks, recurrent neural networks, and deep belief networks.

Model overview

The model utilises three layers, each with 512 RNN units with GRU (gated recurrent units) for short-term memory re- tention. Each layer is densely connected with the adjacent layers to maximise propagation and back propagation. The first layer of this model involves word vector representation, which maps discrete words in a text into a fixed-dimensional feature vector.

The model is divided into two main layers:

1. The model first uses a closed set of bug reports for initial training by processing the bug reports into

word vectors, and then, to focus on layer 1, which involves defect reports and vector representation for whole words, in order to further enhance the confidence of the model, the open bug reports with no developer as- signed are used by extracting information from them in the form of word vectors.

2. The word2vec data processed by the first layer is then fed to the second layer; the function of this layer is to make sense of the vectorized data and find crosslinks between the developers and other bug reports. This layer uses GRU in place of LSTM for short-term mem- ory retention to get a summary of the whole corpus in the form of word vectors.

V. Implementation

Our model primarily makes use of the following technolo- gies:

• word2vec with CBOW(continuous bag of words)

• RNN(recurrent Neural network) in conjunction with GRU(gated recurrent units)

Preprocessing. Getting bug reports from the chromium bug management website was not straight forward, also, there was no way to download a large number of bug reports in bulk. Thus, the only option was to extract the data by look- ing at the requests the browser made while fetching the bug reports, we were then able to extract all the data in bulk by manipulating the requests made to the server and extracting the data.

The data had too many fields like attachment, target, re- view, stability which are not of any concern for bug assign- ment automation, thus we pre-processed the data using the 'jq' utility used for traversing JSON data. The data after pro- cessing was of the form:

{

"id" : 20,

"issue_id" : 98682,

"issue_title" : "Non-responding Windows UNC share hangs bookmark menu",

"reported_time" : "2001-09-07 08:18:34", "owner" : "nobody@mozilla.org",

"description" : "From Bugzilla Helper:\nUser- Agent: Mozilla/5.0 ….

… displayed the bookmark menu directly.", "status" : "RESOLVED",

"resolution" : "WORKSFORME"

},

After processing the dataset, we pre-process the objects im- ported from the JSON data. As we made no use of applica- tion specific fields like hexcode, URLs, stack traces etc, we remove it from our dataset to simplify the results, we then tokenize all the objects extracted and remove the none words and the punctuations present, as the tokenizer does not make any use of it. Both open issues and the closed issues are treated the same as far as preprocessing is concerned(training and testing data should be of same shape and form).

Word Tokenize. After removing the stopwords from the corpus during the preprocessing stage. We used word2vec library for word tokenization as it is able to represent the relationships between words in a low dimensional space in which the words which are closely related are more similar to each other compared to the sparsely related ones.

This approach also overcomes the limitation of the bag of words(BOW) approach and the skip grams approach as it preserves the context information, then in order to extract the vocabulary for word2vec, we make use of the words having frequency more than k. Usually this threshold(k) is set to 10, resulting in sufficiently diverse vocabulary.

Validating results. In Order to best utilize the data we have and to get a reliable estimate of our model's perfor- mance, we cross validated using the 10 fold cross validation approach to determine the results(accuracy of the model on the closed bug report database).

Final Results

Our model was able to achieve a substantial accuracy with the training data, as can be seen by the output of 10 fold cross validation below:

Top10 accuracies for all CVs: [96.24554005838469, 91.16966793325624, 91.02938788764409, 96.05220492866408, 95.22690437601297, 95.74150787075394, 91.23222748815166, 82.29234263465283, 83.13972513089006, 79.65756216877293]
Average top10 accuracy: 90.17870704771835

but the actual accuracy with the novel test dataset took a sig- nificant hit(like in most other papers we reviewed). Which the figure 2. Depicts accurately.
The accuracy of our model in comparison to other models using same or similar dataset can be seen below, we can thus, infer that increasing the number of neurons in the densely linked layers did have an impact on the overall accuracy. Further, our decision to use gated recurrent units instead of LSTM did not seem to have an impact on accuracy, although it did slightly reduce the resources required for computation.

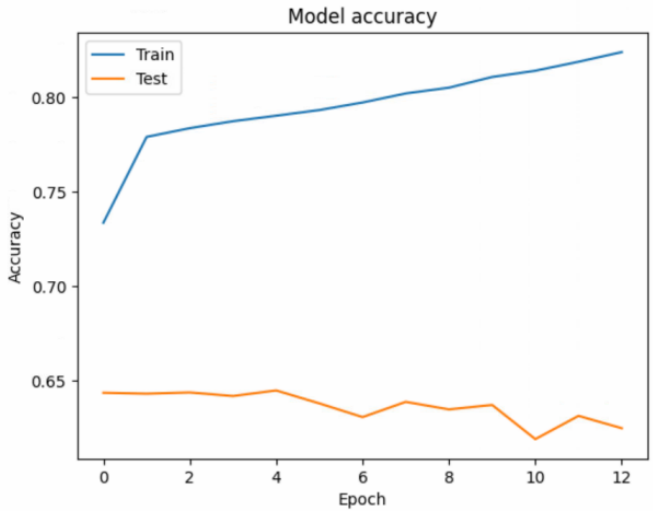

Figure 2. Accuracy values for the best iteration of the mozilla firefox dataset

VI. Conclusion

In this paper we discussed an approach to the problem of bug assignment to developers, one solution of which was proposed by [1], we work upon this solution to further en- hance the accuracy of the model.
The major modifications made by us include :
1. Using gated recurrent units in place of LSTM for faster execution.
2. Increasing the number of neurons in the densely linked layers to increase the performance ceiling of the model.

Table 1. Accuracies compared across other approaches

Classifiers	CV-1	CV-2	CV-3	CV-4	CV-5	CV-6	CV-7	CV-8	CV-9	CV-10	Average
MNB + BOW	22	23	24	26	29	32	34	36	39	38	30.4
Cosine + BOW	18	22	25	28	29	31	34	36	37	38	29.8
SVM + BOW	19	17	15	18	21	19	20	22	23	22	19.6
Softmax + BOW	17	13	13	14	12	12	12	12	13	13	13.1
Softmax + DBRNNA	39	37	40	44	45	47	51	53	54	56	46.6
This Model	51	43	44	42	48	46	55	55	53	59	49.6

The GRUs short term memory function did not seem to make a noticeable effect on the performance of our model, although the computation complexity was considerably re- duces.
Further, the increased number of densely linked neurons were able to achieve slightly better performance, so much so that it outpaced our reference model, few times we ran the model. the one mentioned in the table 1 is the best we were able to achieve through our ten fold cross validation method

VII. REFERENCES

[1]. S. Mani, A. Sankaran, and R. Aralikatte, Deeptriage: Exploring the effectiveness of deep learning for bug triaging, Corr, vol. abs/1801.01275, 2018, Available: http://arxiv.org/abs/1801.01275

[2]. R. R. Panda and N. K. Nagwani, Classification and in- tuitionistic fuzzy set based software bug triaging tech- niques, Journal of king saud university - computer and information sciences, vol. 34, no. 8, Part B, pp. 63036323, 2022, doi: https://doi.org/https://doi.org/10.1016/j.jksuci.2022.01.020.

[3]. A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, Fuzzy set and cache-based approach for bug triaging, in Proceedings of the 19th acm sigsoft sympo- sium and the 13th european conference on foundations of software engineering, in Esec/fse 11. Szeged, Hun- gary: Association for Computing Machinery, 2011, pp. 365375. doi: 10.1145/2025113.2025163.

[4]. D. Cubranic and G. C. Murphy, Automatic bug triage using text categorization, in International conference on software engineering and knowledge engineering, 2004.

[5]. J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, Auto- matic bug triage using semi-supervised text classifica- tion, Arxiv, vol. abs/1704.04769, 2017.

[6]. X. Xia, D. Lo, X. Wang, and B. Zhou, Accurate de- veloper recommendation for bug resolution, in Pro- ceedings - 20th working conference on reverse engi- neering, wcre 2013, in Proceedings - working con- ference on reverse engineering, wcre. United States of America: IEEE, Institute of Electrical and Elec- tronics Engineers, Dec. 2013, pp. 7281. doi: 10.1109/WCRE.2013.6671282.

[7]. G. Jeong, S. Kim, and T. Zimmermann, Improving bug triage with bug tossing graphs, in Esec/fse 09, 2009.

[8]. P. Bhattacharya and I. Neamtiu, Fine-grained incre- mental learning and multi-feature tossing graphs to im- prove bug triaging, 2010 ieee international conference on software maintenance, pp. 110, 2010.

[9]. W. Wu, W. Zhang, Y. Yang, and Q. Wang, Drex: Devel- oper recommendation with k-nearest-neighbor search and expertise ranking, 2011 18th asia-pacific software engineering conference, pp. 389396, 2011.

[10]. D. Matter, A. Kuhn, and O. Nierstrasz, Assigning bug reports using a vocabulary-based expertise model of developers, in 2009 6th ieee international working conference on mining software repositories, 2009, pp. 131140. doi: 10.1109/MSR.2009.5069491.

[11]. F. Servant and J. A. Jones, Whosefault: Automatic developer-to-fault assignment through fault localiza- tion, 2012 34th international conference on software engineering (icse), pp. 3646, 2012.

Cite this article as :

Aryan Singh, Deepanshu Singhaniya, Dr. Ruchika Malhotra, "Evaluating Deep Learning Methods For Classifying Bugs", International Journal of Scientific Research in Science and Technology (IJSRST), Online ISSN : 2395-602X, Print ISSN : 2395-6011, Volume 10 Issue 3, pp. 29-35, May-June 2023. Available at doi : https://doi.org/10.32628/IJSRST523102143
Journal URL : https://ijsrst.com/IJSRST523102143