TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 18, No. 6, December 2020, pp. 2977~2982
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v18i6.16743

Harnessing deep learning algorithms to predict software refactoring

Mamdouh Alenezi[1], Mohammed Akour[2], Osama Al Qasem[3]
[1]Computer Science Department, Prince Sultan University, Saudi Arabia
[2,3]Information Systems Department, Yarmouk University, Jordan

Article Info

Article history:

Received May 19, 2020
Revised Jun 27, 2020
Accepted Jul 6, 2020

Keywords:

Deep learning algorithms
Measurement
Software maintenance
Software refactoring
Source code analysis

ABSTRACT

During software maintenance, software systems need to be modified by adding or modifying source code. These changes are required to fix errors or adopt new requirements raised by stakeholders or market place. Identifying the targeted piece of code for refactoring purposes is considered a real challenge for software developers. The whole process of refactoring mainly relies on software developers' skills and intuition. In this paper, a deep learning algorithm is used to develop a refactoring prediction model for highlighting the classes that require refactoring. More specifically, the gated recurrent unit algorithm is used with proposed pre-processing steps for refactoring prediction at the class level. The effectiveness of the proposed model is evaluated using a very common dataset of 7 open source java projects. The experiments are conducted before and after balancing the dataset to investigate the influence of data sampling on the performance of the prediction model. The experimental analysis reveals a promising result in the field of code refactoring prediction.

Corresponding Author:

Mamdouh Alenezi,
Computer Science Department,
Prince Sultan University,
Riyadh 11586, Saudi Arabia.
Email: malenezi@psu.edu.sa

1. INTRODUCTION

Software applications are endlessly maintained and modify to add new requirements, fix errors, or adapt new modules. Requirements continuously change as the market place is susceptible to stakeholders' demands. Therefore, software applications should be evolved continuously to make sure the stakeholders are satisfied. During software maintenance, programmers are asked to add a new feature, remove and/or modify existing ones. The process of adapting these features needs to modify the software systems to meet the required requirement, this process called refactoring [1]. Code refactoring plays an important role in enhancing software quality by evolving the internal structure without affecting the intended behavior [2].

Extensive research has been conducted for addressing the relationship between refactoring software systems and the quality measurements [3-10]. All experimental results show how refactoring has a direct influence on improving software quality. Therefore, predicting refactoring promptly should be investigating, and building an accurate model becomes mandatory. Software developers still face a real challenge to pick the right time and software code for refactoring purposes as the operation needs time and budget [11]. Therefore, developers should be sure about which piece of code should be evolved before starting the process of refactoring to adopt the new requirements. Although developer experience shapes the most successful factor in this process, still prediction algorithms might be a helpful tool in this matter. These algorithms can provide the developers with some insights about which part of code should be refactored and when.

Different methodologies are designed and built to help developers in the refactoring process such as code smells detection strategies [12], logic meta-programming [13], invariant mining [14] and search-based [15, 16]. Moreover, machine learning is harnessed in the area of prediction and shows noticeable performance in terms of prediction in various fields as computer vision, defect prediction, natural language processing, code comprehension, bioinformatics, speech recognition, and finance [17-24]. Several machine learning algorithms are utilized in code refactoring prediction at class and method level as well [25, 26].

The main contribution of this work is investigating the effectiveness of deep learning algorithms in building refactoring prediction models at the class-level. The implemented deep learning algorithm is gated recurrent unit (GRU).To the best of our knowledge, this algorithm is used for the first time for refactoring prediction at the class level. In this work 7 open-source Java-based projects are used to assess the effectiveness of the studied algorithm.

2. LITERATURE REVIEW

There are several attempts in the literature to use machine learning to predict and suggest refactoring. Amal *et al.* [27] used search-based software engineering for software refactoring. They used an artificial neural network (ANN) and genetic algorithms (GA) to choose between refactoring solutions. They used the opinion of 16 different software engineers to manually evaluate the refactoring solutions for training. They developed a predictive model to evaluate the refactoring solutions for the remaining iterations. The approach outperformed the manual refactoring approach. Kumar *et al.* [28] used 25 source code metrics at the method level to predict the need for refactoring. They used a publicly available dataset of five open-sourced software systems to investigate the performance of ten different machine learning classifiers. They used three different data sampling methods to tackle the unbalanced data issues.

Al Dallal [29] discussed a measure and a model to predict whether the method(s) in a class needing move method refactoring and achieved a prediction accuracy of more than 90%. The author built the predictive model over seven object-oriented systems. Aniche *et al.* [30] investigated the effectiveness of six different machine learning algorithms (logistic regression, naive Bayes, support vector machine, decision trees, random forest, and neural network) in predicting software refactoring. They used a dataset consists of over two million refactorings from 11,149 real-world projects. Pantiuchina *et al.* [31] proposed an approach to prevent instead of fix code quality issues that predict code smells. The approach uses source code quality to predict whether a module is likely to be affected by code smells in the future. The topic at hand is still in its infancy. Several advances can be made to predict code smells and refactoring opportunities that would eventually improve software quality and maintenance.

3. RESEARCH METHODOLOGY

This section presents the proposed approach for Software refactoring prediction. The proposed approach is divided into two main stages. The first stage performs a set of necessary pre-processing procedures on datasets. In the second stage, the deep learning algorithm is applied to the datasets to predict the need for refactoring or not by using gated recurrent unit (GRU) algorithm. The structure of the proposed approach is outlined in Figure 1. More details of the approach are given in the next subsections.

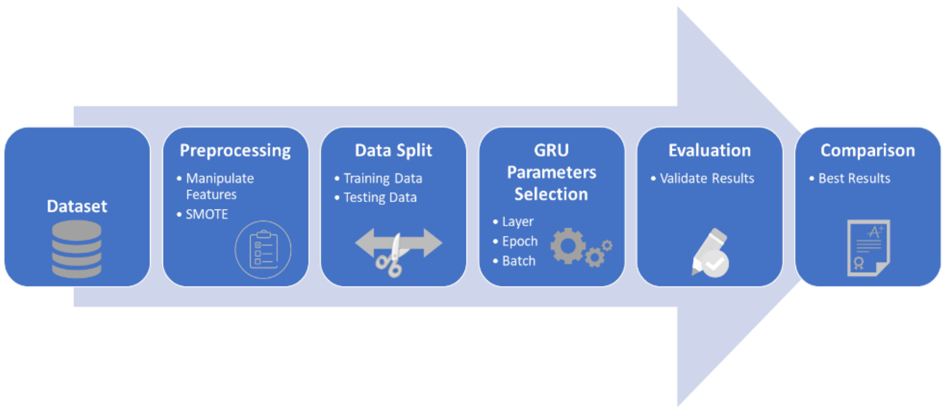

Figure 1. Proposed methodology

3.1. Datasets

In this paper, we used a public empirical dataset containing refactoring data for 7 open-source systems (antlr4, junit, mapdb, mcMMO, mct, oryx, titan) [32]. The experimental dataset used in our study is freely and publicly available at the PROMISE Repository. This makes our work easily reproducible. The dataset used in our experiments is manually validated by creating the source code metrics and the refactoring dataset for two subsequent releases of 7 well-known open-source software Java applications. The tools used to create this dataset are the RefFinder tool for identifying refactoring in the source code between two subsequent releases and the SourceMeter tool to compute source code metrics. These datasets used for empirical investigations on source code refactoring. The features include source code metrics, the refactoring types, and the relative maintainability index (RMI). There are 23 refactoring types at the class level. The dataset's characteristics are presented in Table 1.

Table 1. Datasets characteristics

Dataset	No. of Attribute	Instances	No. refactoring	Percentage
antlr4	134	436	23	5.2%
junit	134	657	9	1.3%
mapdb	134	439	4	0.9%
mcMMO	134	301	3	0.99%
mct	134	2162	15	0.69%
oryx	134	536	15	2.7%
titan	134	1486	13	0.87%

In the first stage, data preprocessing in this study can be summarized as follows:

– Deletion of unnecessary features: remove the following feature (name, path, LongName, Parent, Component)
– Delete refactoring type feature: 23 refactoring types at a class level have been deleted
– Replacement the summation of refactoring feature: We have modified the value of (summation of refactoring types) in each instance to binary values and encoded the summation of refactoring types "more than one" as (1) and "zero value" as (0). then use this feature as a class label.
– Sampling datasets: To improve the prediction of the minority class should be correct the imbalance problem. For that, we used the synthetic minority over-sampling technique (SMOTE). SMOTE deals with the problem of imbalanced distribution, producing new instances based on K-nearest neighbor (KNN). Computing the KNN value forms based on the similarity (we consider in this paper K = 5).

3.2. Gated recurrent unit algorithm

GRU model is a powerful deep learning model proposed by [33] also introduced in [34], GRU is one kind of the gated RNNs which are used to solve the common problems of gradient vanishing of traditional RNN [35]. GRU contains two gates that use it to control the information flow from the through the network. First, the gate to control the information flows into memory known as an update gate. Second, is to control the information that flows out of memory known as reset gate unlike long short-term memory (LSTM), GRU hasn't had separate memory cell, instead of that gating unit that controls the flow of information inside the unit [36].

3.4. Structural parameter selection for gated recurrent unit

To get an effective GRU model, we need to set key structural parameters, which are: 1) the number of hidden layers, 2) the number of epoch, and 3) batch size. To simplify the GRU model, we choose three hidden layers, which is a general configuration followed by [37], and we utilize for the number of the epoch from 10 until 2500. For the batch size use it between 2 to 15.

3.5. Comparisons

The last step in the proposed methodology is the comparison study, where the balanced data set and unbalanced dataset result are measured and compared.

4. EXPERIMENTAL RESULTS

In this section, we empirically evaluate the performance of the GRU model by using the 7 open-source software systems. We optimized GRU with three hidden layers, with a sigmoid activation function for the output layer. This study is carried out on 7 datasets that belong to different domains. The performance with the unbalanced dataset (experimental scenario 1) and with balanced datasets (experimental scenarios 2) are collected and compared in the next subsections. The performance of the GRU model is evaluated based on accuracy, F-measure, recall, and precision. First, we conducted experiments where the dataset is unbalanced; we have evaluated the performance of the studied algorithm on 7 datasets. Then we repeat the same experiment after applying SMOTE to produce a balanced dataset. Table two summarizes the obtained Performance Results before and after applying SMOTE.

In this section, we report the comparison results of prediction performance on the balanced and unbalanced datasets. The results of scenario 2 experiment that uses the balanced dataset show better results in prediction. Our goal here is to investigate whether the balanced datasets can enhance the refactoring prediction performance using the studied deep learning algorithm on these datasets. Table 2 shows the prediction results with unbalanced data. Table 3 reveals how balancing the data can improve the prediction performance results. Most measurement results are increased noticeably, especially for recall, precision, and F-measures. In comparison between unbalanced and balanced data, the F-measure improves by at least double which is very significant.

Table 2. The results with unbalanced data

Dataset	Accuracy	Recall	Precision	F-measure
Antlr4	84.1	40.0	19.04	25.8
Junit	98.61	33.33	50.0	40.0
MapDB	97.8	50.0	33.3	40.0
McMMO	96.0	50.0	25.0	33.3
MCT	98.4	20.0	12.5	15.3
Titan	99.3	33.3	50.0	40.0
Oryx	97.2	20.0	50.0	28.5

Table 3. The results with balanced data

Dataset	Accuracy	Recall	Precision	F-measure
Antlr4	91.9	100	86.1	92.5
Junit	98.6	100	97.2	98.6
MapDB	99.3	100	98.6	99.3
McMMO	99	100	98	99
MCT	99.8	100	97.8	98.8
Titan	99.3	100	98.7	99.3
Oryx	99.3	100	98.7	99.3

5. CONCLUSION AND FUTURE WORK

This work investigates the effectiveness of using deep learning algorithms in refactoring prediction. Gated recurrent unit (GRU) algorithm is used in this study and the performance is evaluated on 7 open-source software products dataset. Moreover, balancing the data set as an enhancement preprocessing stage is addressed in this study as well. The synthetic minority over-sampling technique (SMOTE) is used for balancing the dataset. To the best of our knowledge, gated recurrent unit (GRU) algorithm is used for the first time for refactoring prediction at the class level. The algorithm shows promising performance results. The experimental results show how a balanced dataset enhances the prediction performance noticeably, where all used measures in this study are increased after using a balanced data set in the experiments. As future work, authors will try to predict the refactoring type at class or method level using a deep learning algorithm.

REFERENCES

[1] A. S. Nyamawe, H. Liu, N. Niu, Q. Umer and Z. Niu, "Automated Recommendation of Software Refactorings Based on Feature Requests," *2019 IEEE 27th International Requirements Engineering Conference (RE),* Jeju Island, Korea (South), pp. 187-198, 2019.

[2] A. Qusef, M. O. Elish and D. Binkley, "An Exploratory Study of the Relationship Between Software Test Smells and Fault-Proneness," *IEEE Access*, vol. 7, pp. 139526-139536, 2019.

[3] E. A. AlOmar, M. W. Mkaouer, A. Ouni, and M. Kessentini, "Do design metrics capture developers perception of quality? an empirical study on self-affirmed refactoring activities," *Proceedings of the 13th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2019),* pp. 300-311, 2019.

[4] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," *International Conference on Software Maintenance, Proceedings*, pp. 576-585, 2002.

[5] E. A. AlOmar, M. W. Mkaouer, A. Ouni and M. Kessentini, "On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics," *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM),* Porto de Galinhas, Recife, Brazil, pp. 1-11, 2019.

[6] M. Iammarino, F. Zampetti, L. Aversano and M. Di Penta, "Self-Admitted Technical Debt Removal and Refactoring Actions: Co-Occurrence or More?," *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, pp. 186-190, 2019.
[7] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Transactions on software engineering,* vol. 30, no. 2, pp. 126-139, 2004.
[8] M. Akour, and M. Alenezi. "Test suites effectiveness evolution in open source systems: empirical study," *Indonesian Journal of Electrical Engineering and Computer Science,* vol. 19, no. 2, pp. 1085-1092, 2020.
[9] M. Alenezi,, M. Akour, & H. Al Sghaier, "The Impact of Co-evolution of Code Production and Test Suites through Software Releases in Open Source Software Systems," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 1, pp. 2737-2739, 2019.
[10] M. Alenezi,, M. Akour, A.Hussien, and M. Z. Al-Saad, "Test suite effectiveness: an indicator for open source software quality," *2016 2nd International Conference on Open Source Software Computing,* pp. 1-5, 2016.
[11] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pp. 50, 2012.
[12] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," *20th IEEE International Conference on Software Maintenance*, *Proceedings IEEE*, pp. 350-359, 2004.
[13] T. Tourw´e and T. Mens, "Identifying refactoring opportunities using logic meta programming," *Seventh European Conference onSoftware Maintenance and Reengineering*, pp. 91-100, 2003.
[14] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," *International Conference on Software Maintenance, Proceedings*, pp. 576-585, 2002.
[15] T. Mariani and S. R. Vergilio, "A systematic review on search-based refactoring," *Information and Software Technology*, vol. 83, pp. 14-34, 2017.
[16] M. O'Keeffe and M. O. Cinn´eide, "Search-based refactoring for software maintenance," *Journal of Systems and Software*, vol. 81, no. 4, pp. 502-516, 2008.
[17] O. Al Qasem, M.Akour, "Software Fault Prediction Using Deep Learning Algorithms," *International Journal of Open Source Software and Processes,* vol. 10, pp. 1-19, 2019.
[18] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam and J. Grundy, "An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models," *IEEE Transactions on Software Engineering*, pp. 1-1, 2020.
[19] K. Liu, D. Kim, T. F. Bissyand´e, T. Kim, K. Kim, A. Koyuncu, S. Kim, and Y. L. Traon, "Learning to spot and refactor inconsistent method names," *Proceedings of the 41st International Conference on Software Engineering*, pp. 1-12, 2019.
[20] H. Alsghaier, M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier," *Software: Practice and Experience,* vol. 50, no. 4, pp. 407-27.2020.
[21] M. Akour, H. Alsghaier, and O. Al Qasem, "The effectiveness of using deep learning algorithms in predicting students achievements," *Indonesian J. Elect. Eng. Comput. Sci.,* vol. 19, no. 1, pp. 387-393.2020.
[22] M. Akour, & O. Al Qasem, H. Al Sghaier, and K. Al-Radaideh, "The effectiveness of using deep learning algorithms in predicting daily activities," *International Journal of Advanced Trends in Computer Science and Engineering,* vol. 8, no. 5, pp. 2231-35, 2019.
[23] M. Akour, and W. Yahya Melhem, "Software Defect Prediction Using Genetic Programming and Neural Networks," *International Journal of Open Source Software and Processes,* vol. 8, no. 4, pp. 32-51, 2017.
[24] M. Akour, S. Banitaan, H. Al Sghaier, and K. Al-Radaideh, "Predicting Daily Activities Effectiveness Using Base-level and Meta level Classifiers," *2019 7th International Symposium on Digital Forensics and Security,* pp. 1-7, 2019.
[25] M. Aniche, E. Maziero, R. Durelli, and V. Durelli, "The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring," *arXiv preprint arXiv:2001.03338*, 2020.
[26] L. Kumar and A. Sureka, "Application of LSSVM and SMOTE on Seven Open Source Projects for Predicting Refactoring at Class Level," *2017 24th Asia-Pacific Software Engineering Conference* (APSEC), pp. 90-99, 2017.
[27] A. Boukhdhir, M. Kessentini, S. Bechikh, J. Dea, and L. Ben Said, "On the use of machine learning and search-based software engineering for ill-defined fitness function: a case study on software refactoring," *International Symposium on Search Based Software Engineering,* pp. 31-45, 2014.
[28] K. Lov, S.Mouli Satapathy, and L. Bhanu Murthy, "Method Level Refactoring Prediction on Five Open Source Java Projects using Machine Learning Techniques," *Proceedings of the 12th Innovations on Software Engineering Conference (formerly known as India Software Engineering Conference),* pp. 1-10. 2019.
[29] J. Al Dallal, "Predicting move method refactoring opportunities in object-oriented code," *Information and Software Technology,* vol. 92, pp. 105-120, 2017.
[30] M. Aniche, E. Maziero, R. Durelli, and V. Durelli, "The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring," *arXiv preprint arXiv:2001.03338*, 2020.
[31] P. Jevgenija, G. Bavota, M. Tufano, and D. Poshyvanyk, "Towards just-in-time refactoring recommenders," *Proceedings of the 26th Conference on Program Comprehension*, pp. 312-315, 2018.
[32] P. Hegedűs, I. Kádár, R. Ferenc, and T. Gyimóthy, "Empirical evaluation of software maintainability based on a manually validated refactoring dataset," *Information and Software Technology*, vol. 95, pp. 313-327, 2018.
[33] B. cho, D. van Merrienboer, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.
[34] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.
[35] G. Shen, Q. Tan, H. Zhang, P. Zeng, J. Xu, "Deep Learning with Gated Recur rent Unit Networks for Financial Sequence Predi," *Procedia Computer Science*, vol. 131, pp. 895-903, 2018.
[36] Y. Gao, and D. Glowacka, "Deep Gate Recurrent Neural Network," *ArXiv, abs/1604.02910*, 2016.

[37] O. Alqasem,, M. Akour, and M. Alenezi, “The Influence of Deep Learning Algorithms Factors in Software Fault Prediction,” *IEEE Access*, vol. 8, pp. 63945-63960, 2020.

BIOGRAPHIES OF AUTHORS

Mamdouh Alenezi is currently the Dean of Educational Services at Prince Sultan University. Dr. Alenezi received his MS and Ph.D. degrees from DePaul University and North Dakota State University in 2011 and 2014, respectively. He has extensive experience in data mining and machine learning where he applied several data mining techniques to solve several Software Engineering problems. He conducted several research areas and development of predictive models using machine learning to predict fault-prone classes, comprehend source code, and predict the appropriate developer to be assigned to a new bug.

Mohammed Akour is an associate Professor of Software Engineering at Al Yamamah University (YU). He got his Bachelor’s (2006) and Master’s (2008) degree from Yarmouk University in Computer Information Systems with Honor. He joined Yarmouk University as a Lecturer in August 2008 after graduating with his master’s in Computer Information Systems. In August 2009, He left Yarmouk University to pursue his Ph.D. in Software Engineering at North Dakota State University (NDSU). He joined Yarmouk University again in April 2013 after graduating with his Ph.D. in Software Engineering from NDSU with Honor. He serves as Keynote Speaker, Organizer, a Co-chair and publicity Chair for several IEEE conferences, and as ERB for more than 10 ISI indexed prestigious journals. He is a member of the International Association of Engineers (IAENG). Dr. Akour at Yarmouk University served as Head of accreditation and Quality assurance and then was hired as director of computer and Information Center. In 2018, Dr. Akour has been hired as Vice Dean of Student Affairs at Yarmouk University.

Osama Al Qasem got his Master’s degree in Computer Information Systems from the Faculty of Information Technology and Computer Sciences, Yarmouk University. Al Qasem has few publications in the fields of software engineering, big data analytics, and software fault prediction and currently he is working on building strong research in the area of big data.