

Commit Classification Into Maintenance Activities Using Aggregated Semantic Word Embeddings of Software Change Messages

Tjaša Heričko*, Saša Brdnik and Boštjan Šumak

Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška cesta 46, Maribor, Slovenia

Abstract

Every change to a software repository (i.e., commit) is described by a software developer committing the change with a message written in natural language, indicating the purpose of the change. Automatically inferring the change intents of commits helps understand and manage software projects and their development. This paper presents and evaluates an approach leveraging the semantic characteristics of textual descriptions in software change messages to classify commits into adaptive, corrective, and perfective maintenance activities. The approach represents a commit message as a set of vectors, using a word2vec model trained on commits of the top starred GitHub repositories. Each vector corresponds to a semantic embedding of a word in the message. The resulting word embeddings are aggregated to a single vector representation for a commit, using the average, maximum, and term frequency-inverse document frequency weighted mapping. The experimental results revealed that the models based on the proposed features and simple classifiers are promising, outperforming the baseline ZeroR algorithm. Compared to the traditional approach that uses keywords as features, the models of the proposed approach performed better overall, and provided a more accurate prediction of adaptive and corrective maintenance activities. The best-performing model with a mean weighted F1-score of 75.9% used the maximum aggregation method, and was based on the Random Forest classifier.

Keywords

software maintenance, software repositories, commit messages, multi-class classification, natural language processing, word embeddings, word2vec

1. Introduction

Throughout software development, the iterative changes made to a software project's source code are typically tracked in a version control system, e.g., Subversion and Git [1]. They are designed primarily to support collaborative software development by storing the code, maintaining the history of every modification to the code, and simultaneously fostering collaboration and communication in development teams over the lifetime of a software project [1, 2]. Nevertheless, the abundance of the available data from version control systems and the code hosting platforms built on them, e.g., GitHub, presents a valuable resource, from which software research and practice can learn in order to understand and manage software projects better [2, 3].


SQAMIA 2022: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 11–14, 2022, Novi Sad, Serbia

*Corresponding author.

✉ tjasa.hericko@um.si (T. Heričko); sasa.brdnik@um.si (S. Brdnik); bostjan.sumak@um.si (B. Šumak)

🆔 0000-0002-0410-7724 (T. Heričko); 0000-0001-5535-3477 (B. Šumak)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Changes to software repositories are performed for various purposes, including adapting, correcting, or perfecting software. Analyzing and categorizing changes helps understand and support the decision-making process of software practitioners, concerned mainly with optimum resource allocation and software quality assurance [4, 5, 6]. Additionally, it enables software researchers to study software changes [7]. However, the intents are often not well-written and documented explicitly by developers [7, 8, 9], making inferring the change intents of commits a challenging task. Thus, much effort has been dedicated to mining software repositories to obtain software artifacts of committed changes to a repository, e.g., source code, explanatory commit messages, or other metadata, and exploit them to categorize commits automatically.

Existing research has already investigated the use of various natural language processing techniques for extracting relevant features from commit messages, with the objective of classifying commits into maintenance activities, including word-frequency analysis [10, 11], topic models [6, 12], and contextual embeddings [4, 8]; however, the semantics of the textual descriptions in messages are yet to be explored to the fullest extent of their potential. This paper aims to investigate commit messages from the perspective of semantic embeddings, and the usefulness of aggregated embeddings for multi-class commit classification. A word2vec model, trained on 400k commit messages from the top starred Java repositories on GitHub, was used to convert words into 300-dimensional vector representations. Next, the word embeddings were summarized into commit embeddings using three alternative strategies – averaging, maximizing, and term frequency-inverse document frequency (TF-IDF) weighting. An experiment was conducted to evaluate the approach on a balanced cross-project dataset of 1,793 labeled commits grouped into three maintenance categories, i.e., adaptive, corrective, and perfective maintenance. Several classification algorithms were tested for the classification task. In achieving the research objectives, we contribute to the body of literature in the following three ways:

- Firstly, to address the problem of classifying commits into maintenance activities, we present commit embeddings, constructed by aggregating the embeddings of words found in each commit message.
- Secondly, to evaluate the proposed approach, we analyzed classification models trained on different classifiers, and compared the results with the traditional keyword-based approach.
- Thirdly, we highlight opportunities for modifications of the approach that may be addressed in future work, which could improve the approach’s performance, as well as provide additional insight into the nature of software changes.

The remainder of the paper is structured as follows. The relevant related work is reviewed in Section 2. Section 3 explains our proposed approach. The obtained experimental results are presented and discussed in Section 4. In Section 5, threats to internal and external validity are outlined, with employed strategies to alleviate or prevent them. Section 6 concludes the paper by presenting the summarized findings, implications, and future research prospects.

2. Related work

Over the years, various features have been used for the classification of commits. Regardless of the main objective of the classification, the features can be divided based on their origin,

whether they were derived from properties of the source code (utilized in [5, 9, 13, 14]), the commit messages and their metadata (utilized in [6, 8, 11, 15, 16]), or external tools integrated with software repositories, e.g., bug tracking systems (utilized in [17]). Source code-derived features were used by Mariano et al. [13], with a count of the added and deleted code lines, and a count of the changed files as inputs of the classification. Hönel et al. [9] used source code density, a measure of the net size of a commit. Levin and Yehudai [5] targeted code changes as defined by Fluri’s taxonomy. Sabetta and Bezzi [14] treated code as a text document written in natural language. Keywords derived from commit messages were used by Saini and Chahal [15]. Textual parts of issues reported in bug tracking systems were considered for classification by Antoniol et al. [17].

In this work, we focused on the classification of commits, with the objective of categorizing commits into maintenance categories based on features derived from commit messages. Various approaches with similar aims have already been proposed in the field. Mockus and Votta [10] used word frequency analysis and normalization to select relevant keywords from commit messages and used them as features. A similar work by Mauczka et al. [16] also classified commits based on a set of keywords used in the message. Gharbi et al. [11] addressed the classification as a multi-label active learning problem, representing each commit message as a vector of feature values using the TF-IDF technique. Fu et al. [6] utilized domain knowledge of software changes to prepare labeled samples used to build the semi-supervised Latent Dirichlet Allocation model, and Yan et al. [12] presented a discriminative probability latent semantic analysis model. Sarwar et al. [8] employed transfer learning based on a fine-tuned neural network, i.e., DistilBERT. Apart from only using commit messages, some existing work relevant to ours has performed classification using both commit message- and code-derived features. In such a setting, Ghadhab et al. [4] employed a pre-trained neural language model, DistilBERT, on commit messages, together with extracted fine-grained code changes. The closest to our work are studies by Ghadhab et al. [4] and Sarwar et al. [8], which have already showcased the usefulness of semantic embedding models for dealing with the unstructured nature of commit messages written in natural language and classifying commit messages. We built on these works by providing an investigation of the usefulness of code embeddings constructed from a self-trained word2vec model, and an evaluation of different aggregation methods to aggregate individual vector representations of words into one embedding for a commit.

3. Approach

A high-level overview of the approach is illustrated in Figure 1. Each step is presented in more detail in the following subsections.

3.1. Dataset

An extended dataset made available by Ghadhab et al. [4] was used in our work. It combines data from three sources, i.e., [5], [16], and [18]. Altogether, the dataset contains commits collected from 109 open-source projects written in Java, covering a wide range of application domains, including integration frameworks, utility libraries, and integrated development environments.

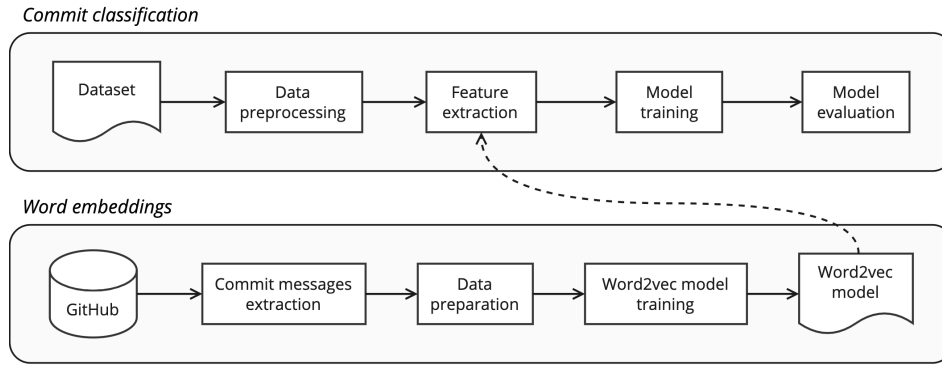


Figure 1: An overview of our approach to commit classification

It consists of 1,793 commits, labeled with maintenance activities belonging to one of the following groups: adaptive ($N=590$), corrective ($N=603$), and perfective ($N=600$). The maintenance categories used, each describing the purpose of a software change, were interpreted as proposed by Swanson [19]. Adaptive maintenance refers to adapting the software to changes in the environment, corrective maintenance refers to activities related to fixing failures, and perfective maintenance refers to improving the software’s performance and quality.

3.2. Data preprocessing

The data preparation phase included multiple steps. The text was first transformed for each commit message to lowercase letters, then numbers and punctuations were removed. Stop-word removal was conducted to eliminate commonly used words with little meaning (e.g., is, not, to). Lemmatization was used to remove the inflectional endings and return only the lemma, the dictionary form of every word.

3.3. Word2vec model training

Word2vec is a well-known and commonly used word embedding learning method in natural language processing presented by Mikolov et al. [20]. The method uses a neural network that can represent a word in a text corpus as a vector with semantic relations to other words in a way that similar words are in close proximity to each other in the vector space. In this work, we trained our own word2vec model. First, 4,000 top starred GitHub repositories based on the Java programming language were extracted using the GitHub Search API. The list contained popular projects, including Elasticsearch, Selenium, Kafka, Gson, and Jenkins. Next, for every repository, a list of all commits, together with the commit subjects, was retrieved by cloning repositories with the `git clone` command and extracting lists of commits using the `git log` command. Commits with empty subjects were removed. From the remainder, 100 commits were selected randomly for each included project. If a project did not contain 100 commits it was removed, and the next available top starred repository after the collected ones was gathered using the GitHub API. The process was repeated until 100 commits were collected from 4,000 projects. The data were preprocessed after obtaining a list of 400k commits on which a word2vec model can learn,

i.e., a text corpus. Lemmatization was performed after transforming the text to lowercase, and removing punctuations, numbers, and stop words. Additionally, short messages with less than six words were removed, as their benefit for training is limited. Duplicated commit messages were dropped after the cleaning steps. The resulting output served as an input to the word2vec training process. We decided to use the skip-gram model, which performs well with smaller training sets and provides good accuracy for rare words. The skip-gram model learns in steps, taking one word as a target in each step, trying to predict the words in its context out to some window size. The model then defines the probability of a word appearing in the context given this target word [20]. The model was trained in 30 epochs using a minimum count of 20, a window size of 2, and a vector size of 300. The resulting trained word2vec model consisted of a vocabulary of 3,384 unique words. In Figure 2, the resulting word embeddings are visualized using the UMAP dimensionality reduction method based on the cosine distance metric. The Figure illustrates the semantic relationship on the example of the word *exception*. The most similar words, based on a computed cosine similarity, are *runtimeexception* (0.628), *ioexception* (0.605), *illegalstateexception* (0.601), and *illegalargumentexception* (0.597), which can also be seen in the Figure.

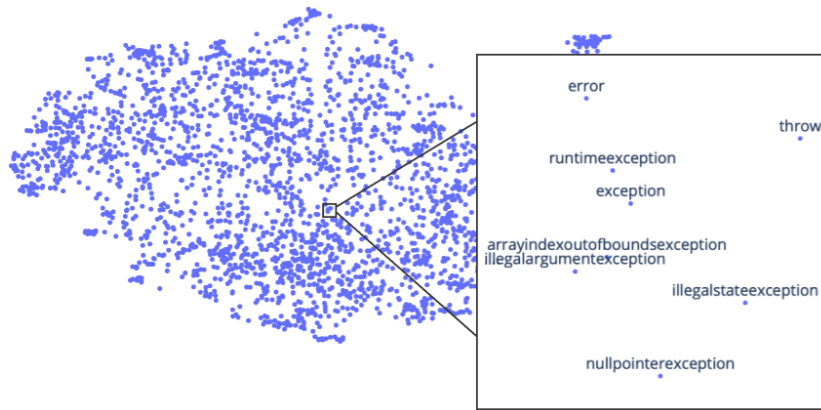


Figure 2: Visualizing word embeddings using UMAP ($n_neighbors=15$, $min_dist=0.1$)

3.4. Feature extraction

For each of the 1,793 commits in the dataset, the learned word2vec model was used to produce an embedding vector associated with each word in the commit message. Words not present in the word2vec model’s vocabulary were not considered. The TF-IDF matrix was calculated next. It assigned a weight to each word in the dataset based on the number of its occurrence in a commit message (i.e., term frequency) and the number of commit messages containing this word (i.e., document frequency) in the following way:

$$TF-IDF = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \quad (1)$$

where $tf_{i,j}$ is the term frequency of the word i in a commit message j , N is the total number of commit messages in the dataset, and df_i is the document frequency of the word i . The

corresponding weight for a word in a commit message reflects its significance to the rest of our dataset. Three aggregation methods were then used to aggregate feature vectors of individual words of a commit message to one commit embedding. The mean value per dimension of words was calculated for averaging. The highest value per dimension of words was calculated for maximizing. For TF-IDF weighting, each word embedding was multiplied by the TF-IDF value found in the TF-IDF matrix, and the resulting vectors were then aggregated by averaging across dimensions.

3.5. Classification task

A multi-class classification approach was performed, in which each commit was assigned a single label $L \in \{adaptive, corrective, perfective\}$. The inputs to the model were the vector representations of commit messages, constructed as described in the previous subsection ($Embeddings_{Avg}$, $Embeddings_{Max}$, $Embeddings_{TF-IDF}$). All three consisted of a 300-dimensional feature vector. The following classifiers were used, selected due to their inclusion in existing research [4, 5, 9, 11, 13]: Logistic Regression (*LR*), K-Nearest Neighbors (*KNN*), Random Forest (*RF*), Gaussian Naïve Bayes (*GNB*), and Decision Tree (*DT*). For the baseline, the *ZeroR* algorithm was used, whose output is the most frequent group in the dataset – in our case, this was corrective maintenance. Also, the traditional keyword-based approach, which uses a set of 20 keywords extracted from commit messages as binary features – True in the case where the keyword is present in the message, and False if the keyword is not present – was implemented as described in [4], to allow the comparison with our approach. Three times repeated 10-fold cross-validation was used to estimate the performance of the models using the mean accuracy and F1-score, averaged per all repeated folds, as the evaluation metrics.

4. Experiment

4.1. Research questions

To assess the proposed approach and its variations based on the aggregation method used, an experiment was conducted, with the aim of addressing the following two research questions:

- (RQ1) Can the proposed approach classify commits accurately into maintenance activities?
- (RQ2) How does the proposed approach perform compared to the traditional keyword-based approach?

4.2. Results and discussion

First, the models were assessed regarding accuracy – the overall percentage of correct predictions. The accuracy of all the folds across different classifiers is presented in Figure 3. Considering the mean accuracy of folds, when *LR* and *RF* classifiers were used, all three variations of the proposed approach outperformed the traditional approach. Only in the case of *KNN* and *GNB*, the traditional approach performed slightly better than the model using TF-IDF weighted embeddings, and in the case of *DT*, the traditional approach performed better than the models

using averaged and TF-IDF weighted embeddings. Comparing the best classification models of every feature in terms of the mean accuracy, the three models using the feature vectors proposed by this work (72.1%, 75.5%, and 65.6%) outperformed the traditional model (63.7%).

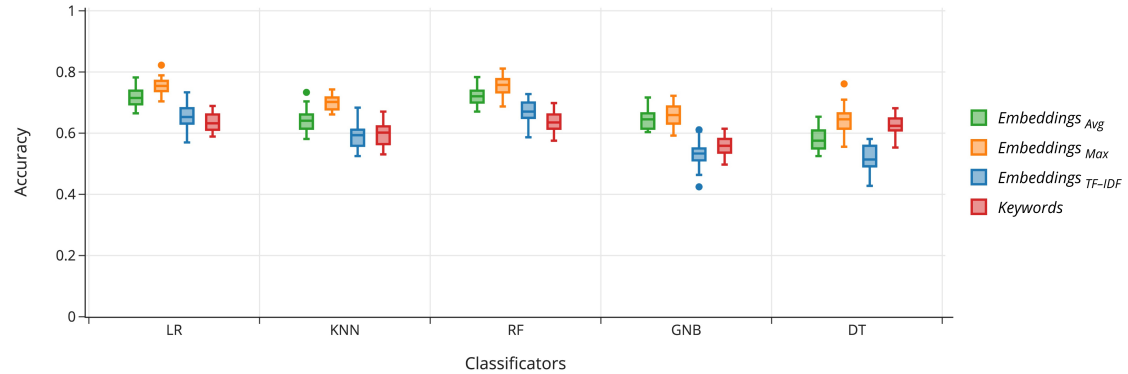


Figure 3: Accuracy of the variants of the proposed approach and traditional approach across different classifiers

Next, as accuracy as an evaluation metric can hinder some problems of the classification models, the F1-score, a weighted average of precision and recall, was also used for assessment. In Table 1, the weighted F1-score is presented, obtained for each model by all the folds and averaged by the total number of folds ($N=30$). When evaluating the performance of the models using the F1-score, the findings are generally similar to the ones observed by the accuracy. However, a larger difference between the mean score of the traditional approach and the mean score of the proposed approach’s variations for the metric F1-score ($\Delta=4.67\%$) compared to the metric accuracy ($\Delta=4.25\%$) hints that the traditional approach has some underlying problems related to false positives and false negatives. Comparing the best models of every feature in terms of the F1-score, the three models using the feature vector proposed by this work (71.8%, 75.9%, and 67.6%) outperformed the traditional model (64.0%).

Table 1

Weighted F1-score (%) of the variants of the proposed and traditional approach across different classifiers

	LR		KNN		RF		GNB		DT		ZeroR	
	\bar{x}	SD	\bar{x}	SD	\bar{x}	SD	\bar{x}	SD	\bar{x}	SD	\bar{x}	SD
<i>Embeddings_{Avg}</i>	71.7	3.3	64.1	3.5	71.8	2.6	64.3	3.1	58.5	3.6	17.0	3.1
<i>Embeddings_{Max}</i>	75.3	2.5	70.0	2.6	75.9	3.5	66.0	3.4	63.5	3.7		
<i>Embeddings_{TF-IDF}</i>	65.6	3.9	59.3	3.5	67.6	3.3	52.0	4.0	52.6	4.1		
<i>Keywords</i>	64.0	3.1	59.0	3.9	64.0	3.2	52.9	3.6	62.9	3.2		

LR=Logistic Regression, KNN=K-Nearest Neighbors, RF=Random Forest, GNB=Gaussian Naïve Bayes, DT=Decision Tree, \bar{x} =Mean, SD=Standard Deviation

All the proposed models performed better than the baseline *ZeroR* algorithm in terms of accuracy (33.6%) and the F1-score (17.0%). The overall best performing model, with a mean accuracy of 75.5% and weighted F1-score of 75.9%, used maximized embeddings as a feature

vector with *RF* as the classifier. In Figure 4, we attempt to visualize our dataset, represented using the best performing feature vector (maximized word embeddings) by transforming the 300-dimensional vectors to 2-dimensional vectors using the UMAP dimensionality reduction method and grouping commits by the maintenance category. We can observe rough areas and patterns where a particular commit category is prevalent. This visualization hints at the value and appropriateness of why using such representations of commit messages can be useful.



Figure 4: Commit embeddings constructed with maximizing word embeddings visualized using UMAP ($n_neighbors=100$, $min_dist=0.3$)

Performance on a more fine-grained level, i.e., per maintenance category, was observed, to analyze further the differences in performance of the proposed and traditional approaches. The normalized confusion matrices on the example of the four models averaged across all folds using the same classification algorithm are presented in Figure 5. We chose to present the performance results of models built with the *RF* because this classifier provided the best results for our variations of the proposed approach. Although this was not the best performing classifier for the traditional approach, the difference between the best one (which uses *LR*) was small; the difference in the mean accuracy was 0.074%, and the difference in the F1-score was 0.023%. The presented confusion matrices gave us a better insight into whether all maintenance categories were being predicted equally well, and what errors the models were making. By comparing the actual categories with the predicted ones, we can observe that all three variations of the proposed approach had a high percentage of true positive predictions (predicted categories are the same as actual categories), which can be indicated from confusion matrices by the dark-colored diagonals. In addition, all three had a similar degree of true positive rate for all three maintenance categories, meaning that the proposed approach worked equally well for all the maintenance categories. On the other hand, the traditional approach, relying only on a set of predefined keyword features, had a higher rate of misclassifications. It was very successful for predicting perfective commits correctly, yet less successful for adaptive and corrective commits, since the approach predicts a large portion of the commits belonging to the two groups falsely as perfective instead.

In the context of **(RQ1)**, we wanted to evaluate how the proposed approach performed for commit classification into maintenance activities. Our results show that using the proposed features to present software change messages described by the unstructured nature-language text, based on aggregation methods performed on word embeddings, improved the baseline

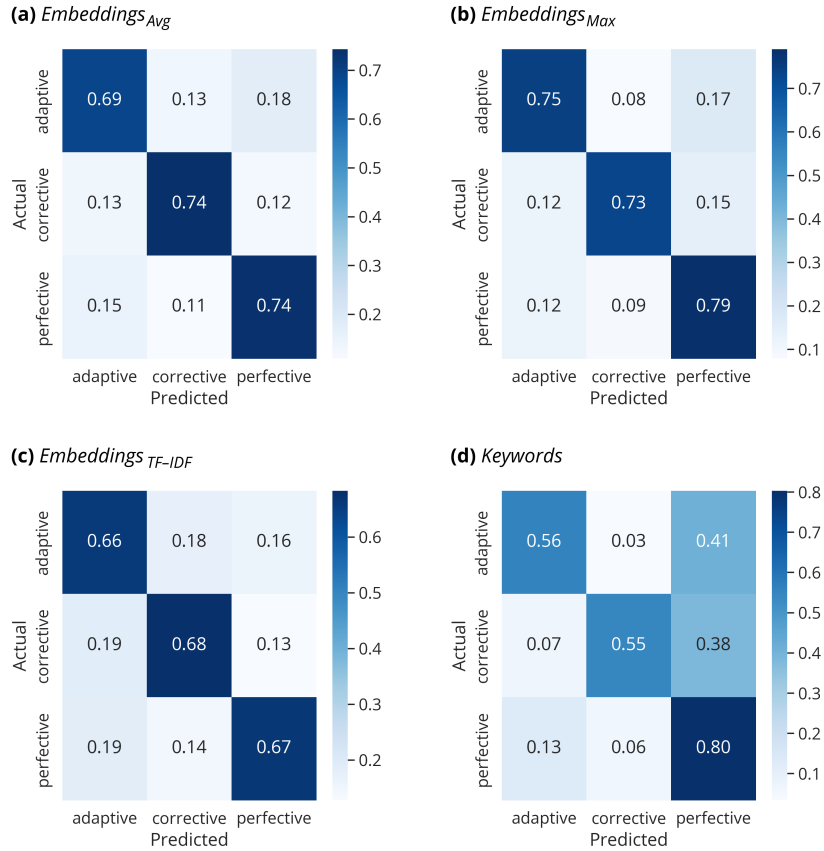


Figure 5: Normalized mean confusion matrices for models trained with the *RF* classifier

algorithm significantly. Such features were shown to contain indicative information to classify commits. The best performing models were the ones using maximizing of word embeddings as an aggregation method.

In the context of (RQ2), we wanted to make our work comparable by evaluating the effectiveness of our approach in comparison to the traditional keyword-based approach. By reproducing the traditional keyword-based approach, we found that, overall, the proposed approach outperformed the traditional approach. Although the proposed approach was less successful at predicting perfective commits, the approach improved the prediction of adaptive and corrective commits compared to the traditional approach.

5. Threats to validity

5.1. Internal validity

The results of this study were largely affected by the self-trained word2vec model embeddings. A major threat to validity relates to the selected subjects used in the model’s learning process. As the learning relies heavily on the quality of commit messages given to the model as inputs,

we chose commits from top starred repositories as a proxy of popular repositories, assuming that commits in popular projects were more likely to be well-documented and messages would correspond adequately to the applied software changes. To ensure further that the included messages would benefit the learning process, short commits with fewer than six words were excluded. To mitigate the risk of capturing semantic characteristics of messages written by only a few developers, we selected commits from a number of different repositories. Despite all these efforts, there is no guarantee that the resulting commits are good representatives for all software projects. This may impact the generated word embeddings used in the classification directly, and, therefore, the classification performance. In addition, it is essential to note that the selected word vector representation method could have affected the classification results. Alternatively, other methods could be used, e.g., GloVe. The parameter values of the word2vec model, e.g., model type, vector dimensions, and the number of epochs, could have also impacted the results. Note that our work did not set out to find optimal values for best results but instead attempted to validate the proposed approach. Our findings were also influenced by the dataset chosen for commit classification. To mitigate threats related to data quality, we reused an existing dataset used previously by several researchers. The validity threats reported by the authors related to the gathering and labeling of the included commits in the dataset should be considered.

5.2. External validity

To mitigate threats related to the generalizability of the resulting word embeddings based on the trained word2vec model, we used a number of commits from various software repositories. It is possible that the results would differ for less popular repositories, closed-source repositories, and repositories of projects written in other programming languages, because the obtained results were restricted to taking into account solely popular open-source repositories of Java code. To maximize the generalization of our findings with regard to the classification task, classification was performed in a cross-project setting. However, the dataset was somewhat limited in size. In addition, the generalization of our findings may not apply to closed-source software and projects written in programming languages other than Java. Repeated k-fold cross-validation was employed to address the risk of under- and overfitting the training data.

6. Conclusions

In this work, we studied the effectiveness of aggregated word2vec embeddings for classifying commits into maintenance categories. We demonstrated that the proposed features, capturing the semantics of commit messages, performed well with simple classifiers, outperforming the *ZeroR* baseline significantly. Additionally, we put the results of the proposed approach in relation to those of the traditional keyword-based approach and highlighted the differences.

The approach is still subject to improvements in future work. Searching for the optimum parameter values of the word2vec model should be performed, for representing words in a semantical vector space adequately. More strategies to assess the quality of commit messages in the data preprocessing step should be considered, for example, dealing with links present in the commit message, as the applied preprocessing steps were found to be insufficient. Future research can focus on additional aggregation methods, e.g., summarization, minimization,

and even concatenation, aiming to find the method that best preserves the relevant semantic information from the commits. Applying strategies to deal with words of commit messages not present in the word2vec vocabulary should be addressed in future work. Hyperparameter tuning can be employed for finding the optimal parameters of classification models. Other classification methods, apart from the ones used in the study, especially neural network-based and ensemble models, could provide better results. In the future, language-agnostic classification models that work for multiple languages should be the focus. Larger and cross-language datasets should be obtained, enabling this research direction. In the current approach, the model can describe the maintenance nature of an entire commit only – each commit can belong to one group only. In the view of some commits belonging to multiple maintenance groups simultaneously, particularly in the case of merge commits, steps should be taken towards multi-label classification.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

References

- [1] N. N. Zolkifli, A. Ngah, A. Deraman, Version control system: A review, *Procedia Computer Science* 135 (2018) 408–415. doi:10.1016/j.procs.2018.08.191.
- [2] G. Gousios, D. Spinellis, Mining software engineering data from github, in: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 501–502. doi:10.1109/ICSE-C.2017.164.
- [3] H. Kagdi, M. L. Collard, J. I. Maletic, A survey and taxonomy of approaches for mining software repositories in the context of software evolution, *Journal of software maintenance and evolution: Research and practice* 19 (2007) 77–131. doi:10.1002/smr.344.
- [4] L. Ghadhab, I. Jenhani, M. W. Mkaouer, M. Ben Messaoud, Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model, *Information and Software Technology* 135 (2021) 106566. doi:10.1016/j.infsof.2021.106566.
- [5] S. Levin, A. Yehudai, Boosting automatic commit classification into maintenance activities by utilizing source code changes, in: *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE*, Association for Computing Machinery, New York, NY, USA, 2017, pp. 97–106. doi:10.1145/3127005.3127016.
- [6] Y. Fu, M. Yan, X. Zhang, L. Xu, D. Yang, J. D. Kymer, Automated classification of software change messages by semi-supervised latent dirichlet allocation, *Information and Software Technology* 57 (2015) 369–377. doi:10.1016/j.infsof.2014.05.017.
- [7] N. Meng, Z. Jiang, H. Zhong, Classifying code commits with convolutional neural networks, in: *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8. doi:10.1109/IJCNN52387.2021.9533534.
- [8] M. U. Sarwar, S. Zafar, M. W. Mkaouer, G. S. Walia, M. Z. Malik, Multi-label classification

- of commit messages using transfer learning, in: 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2020, pp. 37–42. doi:10.1109/ISSREW51248.2020.00034.
- [9] S. Hönel, M. Ericsson, W. Löwe, A. Wingkvist, Using source code density to improve the accuracy of automatic commit classification into maintenance activities, *Journal of Systems and Software* 168 (2020) 110673. doi:10.1016/j.jss.2020.110673.
- [10] Mockus, Votta, Identifying reasons for software changes using historic databases, in: *Proceedings 2000 International Conference on Software Maintenance*, 2000, pp. 120–130. doi:10.1109/ICSM.2000.883028.
- [11] S. Gharbi, M. W. Mkaouer, I. Jenhani, M. B. Messaoud, On the classification of software change messages using multi-label active learning, in: *Proceedings of the 34th ACM/SI-GAPP Symposium on Applied Computing, SAC '19*, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1760–1767. doi:10.1145/3297280.3297452.
- [12] M. Yan, Y. Fu, X. Zhang, D. Yang, L. Xu, J. D. Kymer, Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project, *Journal of Systems and Software* 113 (2016) 296–308. doi:10.1016/j.jss.2015.12.019.
- [13] R. V. R. Mariano, G. E. dos Santos, M. V. de Almeida, W. C. Brandão, Feature changes in source code for commit classification into maintenance activities, in: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019, pp. 515–518. doi:10.1109/ICMLA.2019.00096.
- [14] A. Sabetta, M. Bezzi, A practical approach to the automatic classification of security-relevant commits, in: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 579–582. doi:10.1109/ICSME.2018.00058.
- [15] M. Saini, K. K. Chahal, Change profile analysis of open-source software systems to understand their evolutionary behavior, *Frontiers of Computer Science* 12 (2018) 1105–1124. doi:10.1007/s11704-016-6301-0.
- [16] A. Mauczka, M. Huber, C. Schanes, W. Schramm, M. Bernhart, T. Grechenig, Tracing your maintenance work—a cross-project validation of an automated classification dictionary for commit messages, in: *International Conference on Fundamental Approaches to Software Engineering*, Springer, 2012, pp. 301–315. doi:10.1007/978-3-642-28872-2_21.
- [17] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, Y.-G. Guéhéneuc, Is it a bug or an enhancement? a text-based approach to classify change requests, in: *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, CASCON '08*, Association for Computing Machinery, New York, NY, USA, 2008. doi:10.1145/1463788.1463819.
- [18] E. AlOmar, M. W. Mkaouer, A. Ouni, Can refactoring be self-affirmed? an exploratory study on how developers document their refactoring activities in commit messages, in: *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWor)*, 2019, pp. 51–58. doi:10.1109/IWor.2019.00017.
- [19] E. B. Swanson, The dimensions of maintenance, in: *Proceedings of the 2nd International Conference on Software Engineering, ICSE '76*, IEEE Computer Society Press, Washington, DC, USA, 1976, pp. 492–497. doi:10.5555/800253.807723.
- [20] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013. doi:10.48550/ARXIV.1301.3781.