Feature Changes in Source Code for Commit Classification Into Maintenance Activities

Richard V. R. Mariano*, Geanderson E. dos Santos†, Markos V. de Almeida‡, Wladmir C. Brandão*
*Department of Computer Science, Pontifical Catholic University of Minas Gerais (PUC Minas), Belo Hozizonte, Brazil
†Department of Computer Science, Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil
‡Department of Computer Science, University of Alberta, Edmonton, Canada
richard.mariano@sga.pucminas.br, viggiato@ualberta.ca, geanderson@dcc.ufmg.br, wladmir@pucminas.br

***Abstract*—Software maintenance plays an important role during software development and life cycle. Indeed, previous works show that maintenance activities consume most of the software budget. Therefore, understanding how these activities are performed can help software managers to previously plan and allocate resources in projects. Despite previous works, there is still a lack in accurate models to classify developers commits into maintenance activities. In the present article, we propose improvements in a state-of-the-art approach used to classify commits. Particularly, we include three additional features in the classification model and we use XGBoost, a boosting tree learning algorithm, for classification. Experimental results show that our approach outperforms the state-of-the-art baseline achieving more than 77% of accuracy and more than 64% in Kappa metric.**

***Index Terms*—Source code changes, software maintenance, classification model, machine learning**

I. Introduction

Software maintenance is an important stage of any software project, essential for software sustainability during its whole life cycle [1]. Indeed, software maintenance usually consumes most of the project budget [2], [3], [4], [5]. Several approaches for software maintenance have been proposed in the literature [6], [7] and understanding how maintenance activities are performed can help software engineers and practitioners to previously plan and allocate resources to the software project, ultimately reducing uncertainty and improving cost-effectiveness relationship. One way to understand how maintenance activities are performed is classifying the activities based on the history of commits in Version Control Systems (VCS). For this, three categories are largely used by both research community and software industry [7]: the adaptive category that refers to new features added to the system, the corrective category that refers to both functional and non-functional issues, and the perfective category that refers to changes that improve software design.

Previous works that investigate commit classification into maintenance activities are based only in simple text analysis on commits, reporting an average accuracy below 60% when evaluated within an unique project and below 53% when evaluated within several projects [8], [9]. The state-of-the-art (SOTA) approach to classify commits [5] propose the use of Gradient Boosting Machine (GBM) [10], [11] and Random Forest [9], [12] classifiers, taking into account commits and source code changes (e.g., *statement added, method removed*) as classification features. The GBM classifier presents average accuracy of 72% and Kappa coefficient of 57%, while the Random Forest (RF) classifier presents average accuracy of 76% and Kappa coefficient of 63%. Despite many efforts towards finding the best approach for commit classification, there is still a lack of high accurate models. In addition, many possible features regarding commits are not taken into account in the current SOTA approach.

In this article, we propose improvements on the SOTA approach used to classify commits into maintenance activities [5]. First, we include the following information regarding quantitative changes in source code as additional features: total added LOC (lines of code), total deleted LOC, and the number of files changed, both per commit. Second, we use XGBoost as a classifier, in addition to Random Forest. In recent works reported in literature XGBoost outperforms other implementations of boosting learning algorithms [13], [14].

Additionally, we evaluate the proposed improvements by contrasting our proposed approach with the SOTA approach, reporting the accuracy and Kappa metrics. The Kappa metric (Cohen's Kappa coefficient) is particularly important in cases when the classification categories are unbalanced, i.e., when there are much more occurrences of one class in comparison to others. This scenario could mislead to a high accuracy, when, in fact, this results from unbalanced classes. Experimental results show that our proposed approach outperforms the SOTA approach achieving 77.32% of accuracy and 64.61% in Kappa metric, with gains of up to 5.13% in accuracy and 6.49% in Kappa metric.

The reminder of this paper is organized as follows. In Section II, we present related work. Section III presents our proposed improvements on the SOTA approach for commit classification. Section IV presents the experimental setup and results. Section V discuss the possible applications of our proposed approach. Finally, Section VI concludes our work and presents directions for future work.

II. Related Work

Many previous works have attempted to propose accurate models to classify commits into maintenance activities. Most of them are based on the commit messages, using text analysis, such as word frequency counting, to find specific relevant

978-1-7281-4550-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICMLA.2019.00096

keywords to be used in classification [7], [15], [16], [9], [4]. For instance, Mockus and Votta [7] proposed a commit comment based model to classify commits, reporting an average accuracy of approximately 60% within the scope of a single project, a large software systems with millions of LOC. Hindle et al. [9] proposed the automation of commit classification by training learning approaches on features extracted from the commit metadata, such as the word distribution of commit messages, commit authors, and modified modules. The authors reported accuracy above 50% and argue that the author's identity of a commit provides much information about the maintenance class of a commit, almost as much as the words of the commit message.

Levin and Yehudai [4] proposed a designated repository mining platform, which was used to create a metric dataset from the top 1,000 highly popular open source GitHub repositories, consisting of 147 million LOC and maintained by over 30,000 software developers. The metrics extracted from the dataset were used to predict maintenance activity profiles. The authors show that there is a strong correlation between some metrics with R^2 values achieving up to 0.83. They argued that their results may help project managers to detect anomalies in the development process and to build better development teams. In a more recent work, Levin and Yehudai [5] combined keywords from commit comments and source code changes (e.g., *statement added, method removed*) to classify commits. The authors used GBM and Random Forest as underlying learning algorithms. Their results showed that both algorithms presented higher accuracy than previous baselines, with Random Forest performing better than GBM. Additionally, both algorithms presented regular Kappa metric values.

Previous works that attempt to classify commits into maintenance activities have not taken into account many commit properties, such as the amount of modified LOC. Different from the previous works, we exploit three novel features regarding quantitative changes in source code performed by commits: the number of added LOC, the number of deleted LOC, and the number of files changed. In addition, we XGBoost as replacement for GBM, since XGBoost outperforms other implementations of boosting algorithms, usually with better accuracy [13], [14].

III. The Commit Classification Approach

This section presents the three steps that we follow to propose and evaluate improvements on the SOTA approach for commit classification into maintenance activities. In addition, Section III-A presents the labeled commit dataset and Section III-B presents the additional features we used for commint classification.

Literature Review: First, we perform a literature review to identify related work on classification of commits into maintenance activities [7], [15], [16], [9], [4] as well as the SOTA approach for this task [5]. The SOTA approach uses the classification procedure proposed by Mockus and Votta [7] to classify their models according to three categories: adaptive, corrective and pefective.

Feature Collection: Second, we collect additional features from GitHub[1] via HTTP requests to the GitHub GRAPHQL API[2]. These features bring more information regarding each commit performed by the developer and potentially improve the classification of commits into maintenance tasks. In Section III-A, we explain how the labeled dataset was obtained and in Section III-B, we explain how the new features can be useful to distinguish commit categories.

Experiments and Analysis: Third, we conduct experiments to evaluate the performance of our proposed improvements on the SOTA approach. For this, we split the labeled dataset in training (85% of the data) and test (15% of the data) and for each classification algorithm we perform cross-validation to tune the hyperparameters. We evaluate the training dataset by using 10-fold cross validation with 5 repetitions, i.e., the 10-fold cross validation was performed 5 times and the average accuracy metric is reported. This procedure provide the best hyperparameters to build the classification model.

A. Labeled Commit Dataset

We use a subset of the GitHub repositories reported in a previous work [5]. The number of stars, forks, dates and sizes were used as a criteria to select 11 repositories, representing a wide domains of software projects, such as IDEs, distributed databases, storage platforms, and integration frameworks. In particular, we select the following repositories: RxJava, Intellij Community Edition, HBase, Drools, Kotlin, Hadoop, Elasticsearch, Restlet, OrientDB, Camel, and Spring Framework.

Additionally, we analyse the commit history of each dataset. For each two subsequent commits c1 and c2 (c2 has been done right after c1), the source code changes between them were identified and registered. These changes were first proposed by [17] that added up to 48 different types of changes. Therefore, for each commit, there are a set of 48 features, one for each type of code change. These features are represented by arrays (of size 48) in which each coordinate corresponds to a change and its value corresponds to the number of times the respective change was performed. Furthermore, for each commit, its comment was inspected and searched for a specific set of 20 keywords. The keywords are also part of the features of the model. They are represented by a binary array (of size 20) where each coordinate corresponds to a keyword and the value "1" indicates the occurrence of that keyword, while "0" indicates the absence.

Thus, we select 68 features (48 + 20), corresponding to source code changes and keyword occurrences, respectively. The labeling process was manually performed. Approximately 100 commits were randomly sampled from the 11 repositories and classified according to one of the three maintenance categories (adaptive, corrective and perfective). When a commit did not present sufficient information to allow classification,

[1]https://github.com/

[2]https://developer.github.com/v4/

another one was selected until finding one which was possible to confidently classify. It is important to note that the unbalance problem was addressed by adding more commits of the starved class from the same project. The final dataset consists of 1,151 manually classified commits.

B. Additional Features

We collect three additional features from the 11 repositories and incorporate them in the labeled commit dataset: the total LOC added by the commit, the total LOC deleted by the commit, and the number of files changed by the commit, giving a total of 71 features in our classification model. The additional features can be extremely useful for separating *adaptive* class from the others, however they may not be conclusive regarding the other classes. As we experimented, added LOC is considerably higher in *adaptive* class, changed files is less significant in corrective, while deleted LOC does not present a strong difference. For this reason, we decided to include both features since their co-occurrences can be helpful to separate the maintenance classes.

IV. Experiments and Results

In this section, we present the experimental setup and results. In particular, in Section IV-A we present the impact of each additional feature in the classification performance, in Sections IV-B and IV-C we present setup procedures used to calibrate the XGBoost and Random Forest classifiers, respectively, and in Section IV-D we present a summary of the results on the comparison between the SOTA and our proposed approach to classify commits into maintenance activities.

A. The Impact of the Features

To evaluate the impact of each proposed feature in the classification task, we evaluate each feature separately and combined using 10-fold cross-validation. Table I shows the results of these evaluation, where NaN refers to add no feature, CF refers to add the the number of files changed by the commit, AL refers to add the total LOC added by the commit, and DL refers to add the total LOC deleted by the commit.

TABLE I: The Impact of Features on Commit Classification

	NaN	CF	AL	DL	CF AL	CF DL	AL DL	CF AL DL
Accuracy	71,69	71,60	72,30	71,62	72,50	71,36	72,66	72,31
Kappa	55,89	55,74	56,99	55,57	57,26	55,23	52,55	56,81

B. XGBoost

The XGBoost was trained and tested in the labeled commit dataset using 10-fold cross-validation. A grid of hyperparameters was passed to the evaluation method in order to find the best ones. The main parameter to tune is the maximum depth of a tree in the XGBoost, since it impacts in the model overfitting. Table II shows an excerpt of some variations in the hyperparameters *colsample_bytree* (subsample ratio of columns) and *max_depth* (maximum tree depth), with 150 iterations. The best observed parameters are in bold.

TABLE II: Hyperparameters grid for XGBoost

max_depth	colsample_bytree	Accuracy	Kappa
1	0.4	0.6952	0.5258
1	0.7	0.7004	0.5334
3	**0.4**	**0.7443**	**0.6011**
3	0.7	0.7280	0.5766
6	0.4	0.7239	0.5683
6	0.7	0.7259	0.5721

From Table II we observe that the best hyperparameters are 0.4 (colsample_bytree) and 3 (for max_depth), since with them the model presented 0.7443 of accuracy and 0.6011 of Kappa metric in training. Using these hyperparameters values, the model was evaluated in the test dataset, presenting an accuracy of 0.7570 and a Kappa metric of 0.6070.

C. Random Forest

Similarly to XGBoost, the Random Forest was trained and tested in the labeled commit dataset using 10-fold cross-validation with a grid of hyperparameters. For the Random Forest, the unique parameter to tune is *mtry*, the number of predictors that are randomly selected for each tree in order to ensure that generated trees are uncorrelated. Using the best *mtry* value of 35 in training, i.e., 35 features are randomly selected among the 70, the Random Forest model was evaluated in the test dataset, presenting an accuracy of 0.7732 and a Kappa metric of 0.6461.

D. Summary of Results

The proposed improvements in the SOTA approach to classify commits into maintenance activities positively impacts in classification accuracy. Table III summarizes the results of our evaluation on SOTA and our proposed improvements (OURS).

TABLE III: Comparison of SOTA and proposed improvements

	SOTA		OURS	
	GBM	RF	XGBoost	RF
Accuracy	0.7200	0.7600	0.7570	0.7732
Kappa	0.5700	0.6300	0.6070	0.6461

From Table III we observe that our approach presents higher accuracy and Kappa coefficient for both classifiers. For XGBoost, our approach outperforms SOTA GBM with gains of 5.13% for accuracy and 6.49% in Kappa metric. Regarding the Random Forest, our approach outperforms SOTA with gains of 1.73% for accuracy and 2.55% in Kappa metric. These results attests that including our proposed features is useful for classifying commits. In addition, using advanced implementations of boosting algorithm, such as XGBoost, indeed can improve classification performance.

V. Applications

The proposed improvements on SOTA approach to classify commits in maintenance activities can produce positive impact in several different software development approaches. In this section, we present some of the potentially applications of the results of the present article.

A previous work [4] investigated the amount of commit that a given developer made in each of the maintenance activities (adaptive, corrective and perfective) and suggested the notion of a developers maintenance profile. The models proposed by the authors to predict developer maintenance profile could be supported by our approach to classify commits into maintenance tasks, and possibly they could yield higher accuracy in their prediction.

Another possible application is the identification of anomalies in the software development process. Managers must keep maintenance activities performed by developers under control, i.e. the number of commits made in each maintenance category. Monitoring unexpected behavior in maintenance tasks and its causes would assist managers to plan ahead and allocate resources in advance. For instance, lower adaptive activity may indicate that the project is not evolving as expected, and lower corrective activity may suggest that developers are neglecting fault fixing. Identifying root causes of such problems may aid the manager to improve projects health. Additionally, recognizing maintenance patterns in successful projects may be useful as guidelines for other projects.

Building a software team is a non-trivial task given the diversity of technological and human aspects [18]. Commit classification may help to build a more reliable and balanced developer team regarding the developer maintenance activity profile [4]. When a team is composed of more developers with a specific profile (e.g., adaptive) than others, the development process may be affected and the ability of the team to meet usual requirements (e.g., developing new features, adhering to quality standards) could also be negatively impacted.

VI. Conclusion

In the present article we proposed improvements on a SOTA approach used to classify commits into software maintenance activities. In particular, we proposed the adoption of three new features and the use of the XGBoost algorithm to perform commit classifications. In addition, we carried out experiments using a labeled commit dataset to evaluate the impact of our proposed improvements on commit classification. Experimental results showed that our proposed approach achieved 77.32% of accuracy and 64.61% of Kappa metric outperforming the SOTA approach with gains of up to 5.13% in accuracy and 6.49% Kappa metric.

As future work, we intend to evaluate other features related to software commits, and use other datasets with a larger number of commits to improved the accuracy of the classifiers. We also intent to evaluate other classification algorithms using different evaluation metrics, e.g., precision and recall, to improve our understanding on the classifiers behavior. Commits have more metadata that were not analyzed in this article due to the scope of the application and intended comparison with the existing approach. Moreover, we intent to use NLP (Natural Language Processing) to investigate the commit text based on the maintenance activities. Thus, we could generate a classification approach able to classify the commits automatically based on the labels discussed in this article. Finally, we intent to use other categories (i.e., activities) proposed in the literature to generalize the results.

Acknowledgment

The present work was carried out with the support of the Coordination of Improvement of Higher Education Personnel - Brazil (CAPES) - Financing Code 001. The authors are also thankful for the support given by CNPq, FAPEMIG and PUC Minas (grant FIP 2019/22461-1S).

References

[1] M. Gupta, "Improving software maintenance using process mining and predictive analytics," in *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution*, ICSME'17, pp. 681–686, 2017.

[2] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of application software maintenance," *Communications of the ACM*, vol. 21, no. 6, pp. 466–471, 1978.

[3] S. Schach, B. Jin, L. Yu, G. Heller, and J. Offutt, "Determining the distribution of maintenance categories: Survey versus measurement," *Empirical Software Engineering*, vol. 8, pp. 351–365, 2003.

[4] S. Levin and A. Yehudai, "Using temporal and semantic developer-level information to predict maintenance activity profiles," in *Proceedings of the 32nd IEEE International Conference on Software Maintenance and Evolution*, ICSME'16, pp. 463–467, 2016.

[5] S. Levin and 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'17, pp. 97–106, 2017.

[6] E. B. Swanson, "The dimensions of maintenance," in *Proceedings of the 2nd International Conference on Software Engineering*, ICSE'76, pp. 492–497, 1976.

[7] A. Mockus and L. G. Votta, "Identifying reasons for software changes using historic databases," in *Proceedings of the 16th IEEE International Conference on Software Maintenance*, ICSM'00, pp. 120–130, 2000.

[8] J. Amor, G. Robles, J. Gonzalez-Barahona, A. Gsyc, J. Carlos, and S. Madrid, "Discriminating development activities in versioning systems: A case study," in *Proceedings of the 2nd International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE'06, 2006.

[9] A. Hindle, D. German, M. Godfrey, and R. Holt, "Automatic classification of large changes into maintenance categories," in *Proceedings of the 17th IEEE International Conference on Program Comprehension*, ICPC'09, pp. 30–39, 2009.

[10] J. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[11] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML'06, pp. 161–168, 2006.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[13] T. Chen and C. Guestrin, "XGBoost : Reliable large-scale tree boosting system," in *Proceedings of the Workshop on Machine Learning Systems at Neural Information Processing Systems*, LearningSys'15, 2015.

[14] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'16, pp. 785–794, 2016.

[15] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *Proceedings of the 19th IEEE International Conference on Software Maintenance*, ICSM'03, pp. 23–32, 2003.

[16] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," *Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005.

[17] B. Fluri and H. C. Gall, "Classifying change types for qualifying change couplings," in *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC'06, pp. 35–45, 2006.

[18] N. Gorla and Y. W. Lam, "Who should work with whom?: Building effective software project teams," *Communications of the ACM*, vol. 47, no. 6, pp. 79–82, 2004.