Commit Classification using Natural Language Processing: Experiments over Labeled Datasets

Geanderson E. dos Santos[0000−0002−7571−6578] and
Eduardo Figueiredo[0000−0002−6004−2718]

Federal University of Minas Gerais, Belo Horizonte MG 31270-901, Brazil

Abstract. Software commits play an important role in collaborative software development. A commit allows developers to collaboratively contribute to open-source software projects. Previous studies show that developers spend a lot of time in identifying and prioritizing critical issues and respective commits. For this reason, understanding what a software commit is trying to solve in the project is very important. In this context, commit classification could help managers to plan and allocate resources in advance for the software project. Despite its relevance, we still lack accurate models to support automated classification of commits into relevant categories. In this work, we conduct an empirical study to evaluate commit classification techniques using the commit message given by developers as input. After an ad-hoc literature review, we created a dataset based on labeled data from different works found in the literature. Second, we compared five Machine Learning (ML) state-or-art algorithms seeking a reliable baseline model. Finally, we applied Natural Language Processing (NLP) to a novel ML model aiming at improving results compared to the baseline models. Our findings show that the new NLP-based model outperforms the random baseline models by a relevant rate, achieving 91% of F-measure for the commit classification task.

Keywords: software commit classification · natural language processing · machine learning.

1 Introduction

Software commits play an important role in the Version Control System (VCS) allowing practitioners to create collaborative source code repositories [7, 9, 22, 23]. These repositories can be shared with the software community, and many developers around the world contribute to important open source projects. The main representative of these VCS platforms is the well known GitHub[1] Web system, which is responsible for storing millions of projects from companies, research groups, and independent developers. The contributors to these systems have an easy-to-use architecture for introducing new features, fixing bugs, testing existing code, and many other capabilities [7, 8]. On the other hand, dealing with online repositories may introduce serious bugs that may not have been properly

[1] https://github.com/

tested [7, 13, 15, 16]. One approach to deal with the probability of introducing bugs into the project is, for instance, to understand what the commit is doing in the source code. Commit classification can, therefore, support the estimation and planning of bug fixes [5, 10, 20] among other uses. In fact, different approaches for commit classification have been studied in the literature [2, 7, 10, 16, 20].

In this paper, we first evaluate five Machine Learning (ML) algorithms used for commit classification aiming to predict the nature of changes. These ML algorithms are used as baseline models and rely on the text provided by the developer who committed the code. One of the advantages of classifying software commits is that it may help practitioners to plan and allocate resources to the project issues. Therefore, planning the software may reduce uncertainty and improve the cost-effectiveness of tasks related to the project development. To classify a software commit, it is necessary to look at the possible inputs a developer may provide in the commit log.

There is no consensus regarding the different types of classification to which a commit refers. Therefore, after an ad-hoc literature review, we collected labeled datasets and created a unified dataset by classifying commits into five categories: corrective, features, non-functional, perfective and unknown. We adopted the definition of Levin and Yehudai [13] for three of the categories: corrective related to fault fixing, perfective based on system improvements, and adaptive corresponding to new features introduction. We also used the definition of Hindle et al. [9], where non-functional is associated with documents and non-functional requirements. In addition, the category unknown is able to aggregate files that are created by VSC systems. For instance, when a merge is executed automatically in GitHub, some files are committed with generic descriptions. Finally, we propose a novel model able to classify commits. Therefore, the final goal of this paper can be stated as follows.

> Propose and evaluate a model based on Natural Language Processing (NLP) to classify software commits.

To provide a mean for comparison between the novel NLP model and baseline models from the literature, we apply all models to a unified dataset of labeled commits. Other works are available for comparison [2, 10]. However, state-of-art work has different categories for software commit classification. For instance, there is a classification of commits in only three categories related to maintenance activities: corrective, adaptive, and perfective [14]. In such work, the authors were able to achieve an accuracy of around 76% with three categories [14]. These labels are very limited for our purpose. So, we opt for a more deep study of the categories, where five classes of commits are taken into account [8]. Using the same five labels adopted for the commit classification, previous work was able to achieve approximately 65% of accuracy from different models [8]. The NLP model we propose, however, was able to achieve expressive results in the labeled dataset, achieving 91% of F-measure for the commit classification problem.

This paper is organized as follows. Section 2 presents the related work. The experimental design and its main phases are discussed in Section 3. The study is divided into 4 steps (i) ad-hoc literature review, (ii) collecting data from labeled datasets, (iii) experiments with the baseline algorithms, and (iv) experiments with a new NLP model to classify software commits. In Section 4, we discuss the main results of this paper. Finally, Section 5 details the threats to validity involved in this research, and Section 6 concludes this paper with insights for future explorations on software commit classification.

2 Background and Related Work

In this section, we present literature work related to the task of software commit classification. The section is divided as follows. We introduce the problem of commit classification (Section 2.1). Next, we discuss in Section 2.2 the tools and algorithms proposed in the literature.

2.1 Commit Classification

The simplest kind of classification is to identify some unknown object as a member of a known class of objects. Classification is a computationally hard problem, and it is the subject of study in different Computer Science areas. One of the many forms of classifying software commits is through labels. The central idea of the labeling mechanism allows users to decide which labels they think are more applicable to the context of their content, in contrast to other classification mechanisms. Many authors have studied how labels are used in these systems [3, 6]. The use of labels has also been extended to Web-based code hosting platforms where tools like issue-trackers (as the popular Bugzilla) offer this functionality.

Previous studies have attempted to propose accurate models to classify software commits. Most works are based on the commit messages, using text analysis, such as word frequency approaches to find specific keywords [8, 9, 13, 17, 22]. For instance, Mockus and Votta [17] have proposed a commit comment based model to classify a software commit. The authors reported an average accuracy of approximately 60% within the scope of a single project (a very large software system known as Firefox with millions of lines of code). In a similar approach, Hindle and colleagues [9] proposed to classify software commits by training Machine Learners on features extracted from the commit metadata, such as the word distribution of a commit message, commit author, and modules modified by the developers. The authors validated the model through 10-fold cross-validation and reported accuracies above 50%. They also pointed out that the identity of the author of a commit provided a lot of information about the subject of a commit, almost as much as the words of the commit message. Furthermore, the authors propose the identification of six categories for commit classification: “perfective”, “merge”, “adaptive”, “feature addition”, “non functional”, and “preventive” [20]. Compared to previous work, our proposed model achieved higher accuracy than these projects in a unified labeled dataset and we used several random models as baselines for comparison.

2.2 Tools and Algorithms for Commit Classification

Rosen and colleagues [20] developed a platform to automatically identifies risky (i.e., bug-inducing) commits and built a prediction model to assess the likelihood of a recent commit introducing a bug in future. The platform is known as *Commit Guru* and allows users to download the data for any project that is processed by the system. Several large open source projects have been successfully processed using *Commit Guru*. Besides this tool, other authors [10] present a platform that generates a set of visualizations to facilitate the analysis of issues in a project depending on their label-based categorization. The tool is named GitHub label analyzer (*GiLa*). They argue the visualizations are useful to see the most popular labels and their respective relationships in a project. The tool also allows identifying the most active community members for those labels and a comparison of the typical issue evolution for each label category. In a different work, Levin and Yehudai [13] built a designated repository mining tool that was used to create a metric dataset. The tool is responsible for processing nearly 1,000 highly popular open-source GitHub repositories, consisting of 147 million lines of code and maintained by over 30,000 developers. The computed metrics were then employed to predict the maintenance activity profiles identified in previous works. The authors show how their results may help project managers to detect anomalies in the software development process and to build better development teams.

A previous work [5] discussed the problem of identifying particular changes that occur across several versions of a program. The authors present a plugin known as *Change Distilling*, a tree differentiation algorithm for fine-grained source code change extraction. Similarly, Casalnuovo and others [2] proposed a tool known as *GitcProc*, a lightweight tool based on regular expressions and source code blocks, which downloads projects and extracts their project history including fine-grained source code information and development time bug fixes. These systems present interesting features for the software commit classification. However, both *Commit Guru* and *GiLa* seem to be in maintenance stage or have been closed by their authors [10, 20]. While *Change Distilling* and *GitcProc* are still available, *Change Distilling* is limited to Java IDE Eclipse. As we do not want to face the same availability problem, we have created a replication repository with all our data and tooling publicly available on GitHub[2].

3 Study Setup

In this section, we present the research questions that guided our study (Section 3.1). Section 3.2 presents the study phases we followed in this work. Section 3.3 presents the dataset used to evaluate all models for the software commit classification problem. Finally, we discuss in Section 3.4 the baselines models applied in this research.

[2] https://github.com/gesteves91/fasttext-commit-classification

3.1 Research Questions

Our experiments aim to answer the following research questions concerning the software commit classification problem using a unified labeled dataset.

RQ1: Are random models effective in the commit classification task?

RQ2: Can we improve the effectiveness of random models for the commit classification task by using natural language processing?

3.2 Study Phases

To answer RQ1 and RQ2, we designed and conducted an empirical study composed of four phases. These phases aim to achieve the final goal of proposing and evaluating a model based on the Natural Language Processing (NLP) to classify commit activities. The phases are described as follows.

Phase 1 Literature Review - An ad-hoc literature review was performed to find related work about software commit classification and identify the state-of-art on labeled datasets, whose features are replicated in this work. Other research projects and tools were analyzed to explore the many possibilities that the software commit classification problem had to offer for our study. At the end of this phase, we were able not only to find important research projects about the subject, but also to set the target in terms of the objective of the study and the datasets used in the experimental phase of this research.

Phase 2 Collecting Data from Labeled Datasets - During the literature review, a search was conducted to find labeled datasets publicly available on the internet for researches to freely test and explore new approaches for software commit classification. We aim to test these datasets against a Natural Language Processing (NLP) algorithm in order to investigate its effectiveness for the target problem. This phase was also responsible for joining the many different datasets into one unified dataset to be used in the experimentation phase.

Phase 3 Experiments with the Baseline Models - In this phase, the experiments and their respective analyses were performed in terms of the selected baseline models. Then, for each algorithm, a cross-validation method was used to test the models [12]. The training dataset was evaluated using repeated 5-fold cross-validation with 5 repetitions, i.e., the 5-fold cross-validation was performed 5 times and the average F-measure metric was reported. Note that the test dataset did not take part in the training process, which provides a more realistic evaluation of the problem. It also prevents the study for overfitting and leading us to incorrect conclusions about the data distribution. All models and their evaluations were implemented in the Python language, with the support of the largely used Scikit-learn package [18]. This phase aims to answer our first research question (RQ1).

Phase 4 Experiments with a new NLP Model - In this last phase, the experiments were performed with a novel NLP model. The average F-measure is reported after a grid search of the hyper-parameters. This phase aimed at comparing results from the proposed NLP model against the baseline models in terms of the target evaluation metric (F-measure). This last phase targets our second research question (RQ2).

To exemplify the experiments described in Phases 3 and 4. Figure 1 illustrates three stages applied to generate the baseline and the NLP model. In Stage 1, we vectorize the words contained in the commit messages using the TF-IDF algorithm. Then, in Stage 2, we tested the performance of five baseline models to classify software commits. Finally, in the last stage, we build a new NLP model based on the fastText algorithm [1] to classify software commit expecting better results than the baseline models and the current literature.

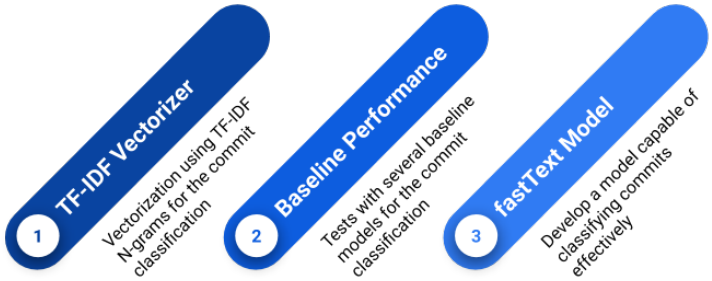

Fig. 1. Phases to create the NLP model to classify software commits.

3.3 Dataset

The dataset collection was gathered from three different sources. These sources are publicly available on the Web under their respective publications [14, 16, 21]. All the provided data were properly labeled using the expertise of experienced developers [14, 16, 21]. Furthermore, the respective authors asked the developers to label their own commits in order to avoid imprecise labels. In total, 5,631 commits were manually classified and unified in our final dataset. The commits are classified following the maintenance activities [14] and the respective authors have applied other methods, such as classifying the similar commits [9].

In order to keep a common version between the datasets and maintaining the nature of the changes, we opted for relabeling the "adaptive" label as "features" since the literature do not agree in this category [9, 14]. Then, the labels we used are "corrective", "features", "non functional", "perfective", and "unknown". Note that even though two of the datasets followed the same classification we used [9], they also classify commits in terms of "merge" and "preventive"

labels. Ignoring these classifications, such as "merge" label [11], is common in the literature. In fact, Version Control Systems (VCS) often provide automatic commit messages for merges, such as "merge commits X and Y", which makes their classification pointless.

Figure 2 shows the distribution of labels in the final dataset. The classes (i.e., labels) are imbalanced in the unified dataset. The "corrective" label is significantly more common than the remaining labels. Nevertheless, the "unknown" label has the lowest rate compared to the other classes. The imbalanced classes are not necessarily a problem for this study as it is discussed in the following sections. Evaluating the F-measure and the correct application of the cross-validation process is enough to deal with the imbalanced data [4].

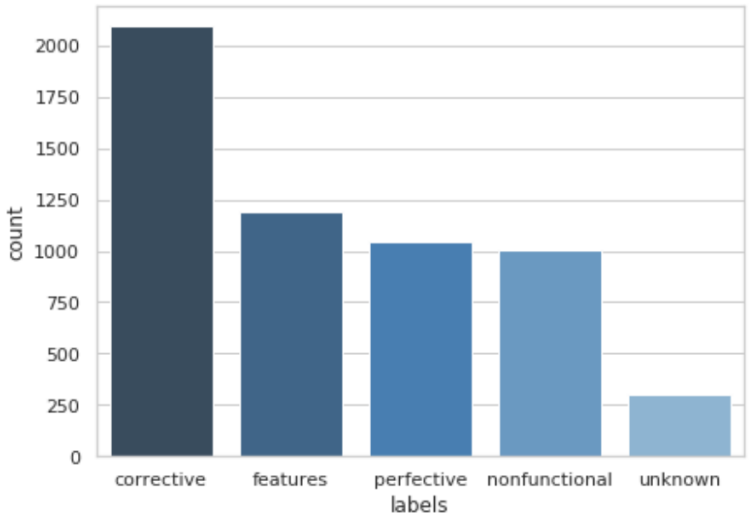

Fig. 2. Distribution of labels x commit messages.

3.4 Baseline and Measures

We considered five machine learning algorithms that are often used for text classification prediction to provide baseline for the comparison [14, 21]: Random Forest (RF), Linear SVC, Multinomial Naive Bayes (NB), Logistic Regression (LR) and Decision Tree (CART). We then evaluate the effectiveness of the considered models using standard F-measure [4].

F-measure is highly necessary for the commit classification problem because, as observed in Figure 2, the labels are not equally distributed in the dataset. In other words, the dataset is fundamentally imbalanced and, then, considering only the accuracy would give imprecise conclusions about the models. F-measure, however, is defined as the weighted harmonic mean of the precision and recall

of the test. Therefore, it considers both precision and recall in its fundamental calculation.

4 Results and Discussion

This section presents our results and discusses the main findings of this study, by answering our two research questions (RQ1 and RQ2).

4.1 Random Models for Commit Classification

The first study we conducted is devoted to answering research question RQ1: "Are random models effective in the commit classification task?". To answer this question, we first create a word vector to represent the sentences in the commit message. The commit message is represented as a string when we access the raw dataset. The word vector representation is responsible for creating the matrix that serves as input for the baseline models. To create the word representation, we opt for the technique known as Term Frequency - Inverse Document Frequency (TF-IDF) [19, 24]. This method calculates values for each word in a document through an inverse proportion of the frequency of the word in a particular document to the percentage of documents the word appears in. For this reason, words with high TF-IDF numbers imply a strong relationship with the document they appear in, suggesting that if a word appears in the query, the document could be of interest to the user. This technique seemed to be a reasonable option to generate the word vectors as discussed in the literature [19, 24].

TF-IDF algorithm can output the N-gram relationship between the words in the corpus. For instance, if the commit message is "my code has a bug", the unigram (or 1-gram where N=1) is "my", "code", "has', "a", "bug". On the other hand, the bigram (or 2-gram where N=2) outputs the following result for the same sentence: "my code", "code has", "has a', "a bug" [19]. Table 1 shows the most frequent unigram and bigram outputted by TF-IDF based on the dataset described in Section 3.3. The results generated by TF-IDF unigrams and bigrams are used for all baseline models of this research. As an example in Table 1, the TF-IDF algorithm found out that the most correlated words to the "corrective" label are "bug" and "fix" for the unigram and "new ui" and "fix bug" for the bigrams. The same structure is followed to output the most correlated words for the remaining labels. The algorithm seemed to find good relationships between the labels and the words. Therefore, the TF-IDF vectors were used as input for the machine learning algorithms.

After generating the word vector representation of the commit messages, we applied random models to test their effectiveness in classifying commits. As explained in Section 3.4, we choose five ML algorithms for this task: Random Forest, Linear SVC, Multinomial Naive Bayes, Logistic Regression (LR) and Decision Tree (CART) [16, 14]. To select a metric to evaluate effectiveness of models

Table 1. Most related 2 words Unigrams and Bigrams.

Label	Unigram	Bigram
Corrrective	bug	new ui
	fix	fix bug
Features	features	new features
	feature	add support
Perfective	design	package structure
	structure	structure refactoring
Non-Functional	improve	improve performance
	performance	improved performance
Unknown	duplicated	duplicate code
	duplicate	remove duplicate

for commit classification, we examined several options, such as F-measure, Accuracy, precision, and recall. Not all metrics would deal well with the imbalanced nature of our dataset. Hence, we opted for the F-measure as the standard metric, as it is the harmonic mean of both precision and recall. Thus, the recall and precision are represented under the F-measure values.

Figure 3 shows F-measure numbers for the five baseline models in the commit dataset. Four of the baseline algorithms have perform relatively well for the task of commit classification (Linear SVC, Decision Tree, LR and Multinomial NB). Only Random Forest does not perform well for the task. The F-measure values for Random Forest achieved only around 20%. The other four algorithms performed relatively well: Multinomial NB (around 73%), Decision Tree (74%), Linear SVC (around 75%), and LR on average 76%. Since LR achieved the best results among the baseline models, it was selected as a representative model in our next study about the effectiveness of commit classification (Section 4.2). Thus, we can conclude that random models have not achieved a great performance for the commit classification problem, as the best algorithm (LR) achieved 76% of F-measure.

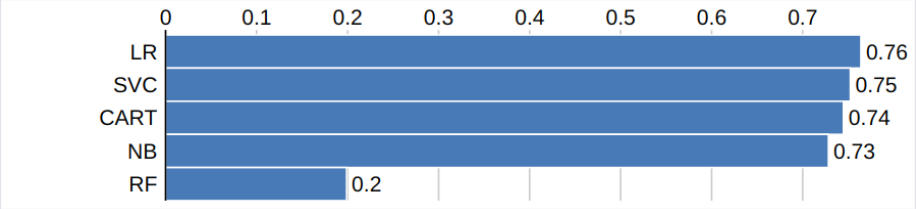

Fig. 3. Baseline ML Models F-measures.

To further support our analysis focusing on LR, we have generated the confusion matrix from the LR execution. Figure 4 exemplifies the errors the LR model

has committed. In the left of Figure 4, we have the actual labels from the unified dataset; in the bottom, we represent the predicted label by the LR algorithm. The total commits correctly classified is represented in the diagonal, where the actual and predicted labels for each commit are situated. As we can see, despite most of the labels are classified correctly (as around 76% where correctly classified), there are commits classified wrongly, especially in the "corrective" label (middle column of the figure). It is important to note that the confusion matrix was tested in 20% of the dataset randomly selected from the entire set.

From the findings presented in this section, we may conclude the following answer to RQ1.

RQ1: Random models are reasonably effective for the commit classification task, since the best models achieved a F-measure of around 75%.

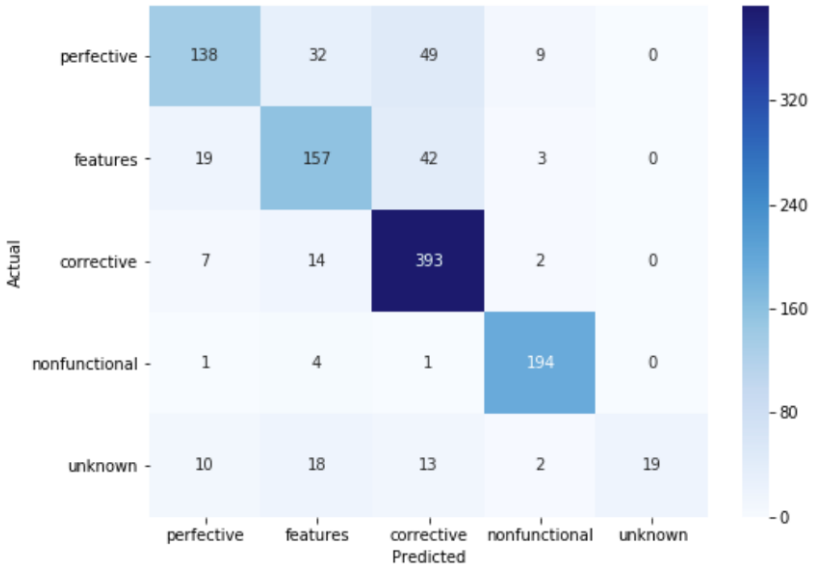

Fig. 4. Confusion Matrix of Actual and Predicted Labels for Logistic Regression.

4.2 Commit Classification using Natural Language Processing

The second experiment is devoted to answering our research question RQ2: "Can we improve the effectiveness of random models for the commit classification task by using natural language processing?". After generating the results shown in the last section, we noticed that Logistic Regression achieved a reasonably acceptable

F-measure for the task of commit classification. However, we also observed an opportunity to improve the results by applying a state-of-the-art NLP algorithm proposed by the Facebook Artificial Intelligence Research [1], named fastText. fastText relies on the skip-gram model, where each word is represented as a bag of character n-grams. Then, the vector representation is associated with each character n-gram, and these words are basically the sum of these representations. It allows the model to compute representations for words that did not appear in the training data. Furthermore, the proponents of fastText also discussed how this approach has made the model execute faster than other algorithms presented in the literature [1]. The fact that fastText algorithm considers words that do not appear on the vocabulary seems important for the commit classification problem because developers are sometimes careless about the commit messages [16]. In other words, it is likely for commit messages to include typos and misspelling words.

Based on this opportunity, we propose the use of fastText to enhance the commit classification task. We evaluated this proposal by replicating our study presented and discussed in Section 4.1 under the same structure and for the same dataset. We have also created a grid search structure for testing the model against different configurations of hyper-parameters. For example, we tested variations of the learning rate and a number of epochs. The best-achieved result is outputted by a learning rate of 0.1 with 25 epochs, where expressively **91%** of F-measure was achieved for the problem of commit classification. We believe the results are high due to fastText nature of dealing well with imbalanced classes [1]. Then, we conclude that fastText was very efficient in the task of commit classification.

Then, we may draw from this experiment the following answer to RQ2.

> **RQ2**: It is possible to more effectively predict software commit by using natural language processing. To do so, we developed a model based on fastText able to achieve 91% of F-measure.

5 Threats to Validity

The study presented in this paper has some limitations that could potentially threaten our results, as we discuss next. The first threat refers to the chosen datasets, the datasets are related to other research projects, and despite they have been labeled for the same purpose of classifying commits into software activities, they do not follow the same fundamental labels. For example, in the maintenance activities dataset, the authors focused on the definition of labels of Levin and Yehudai [14], while in Hindle et al. [9], the authors focused on the definition of commits proposed by the main author. We may not affirm surely in which significance it may interfere in this study results. However, in an ideal circumstance, all the datasets would have the same labels for classification.

Furthermore, we believe the different definitions used by the authors may have impacted in the imbalanced classes of the unified dataset.

Another threat to the validity of our research is related to the features adopted to predict the output using the fastText model. The datasets found in the literature have not considered, for instance, the "preventive" and "merge" labels discussed by Hindle et al. [9]. We could not determine in which extend it may interfere in our results either. However, it is certainly true that our model generated using fastText would not be able to classify commits related to the preventive and merge classes properly. Again, there is no consensus of these labels in the literature. For this reason, it is very difficult to confirm the relevance of these labels to any classification model adopted in further research approaches.

6 Conclusion and Future Work

Software commit classification is extremely relevant for software projects as it allows understanding of whole software life cycle and management. The incorrect introduction of faulty commits is shown to consume most of the project budget. Therefore, understanding how these tasks may be classified is useful for practitioners/managers so that they can plan and allocate resources in advance. State-of-art techniques to classify commits are very limited using a few labels to classify the commits.

In this work, we presented an empirical study to investigate the commit classification based on the commit messages. To do so, we applied a natural language processing approach, considering more labels than the current state-of-art literature about this problem. In this paper, the datasets used from other research projects have five categories: features, corrective, perfective, non-functional, and unknown. In order to compare the baseline models with the natural language processing algorithm, we carried out experiments to evaluate effectiveness of both approaches. First, a hyper-parameter grid was evaluated through repeated 5-fold cross-validation analysis for each algorithm. Then, we evaluated the models in a test dataset, which did not take part in the training phase. We achieved 76% of F-measure by Logistic Regression, which was the best performing baseline model. Moreover, the text representation algorithm (i.e., fastText) was able to reach a high F-measure (approximately 91%). F-measure was chosen for this study because the dataset is imbalanced. Thus, this approach was able to outperform the baseline best performing model; i.e., 76% vs. 91% of F-measure.

As future work, we want to apply the model generated by fastText to other datasets from GitHub or similar tools. Also, the fastText model could eventually be improved with more labeled data from other datasets, or with a practical study to manually labeled data from practitioners. This approach would be similar to the one applied in previous work [14, 16], where the researchers have asked developers to classify their own commits. Furthermore, using our own classified dataset, we could adapt the labels to use all the proposed labels from others [9]; e.g., the preventive and merge labels that were not used in the unified datasets of this research paper. Another possible approach would be the application of

the fastText model to classify commits from participants in a controlled experiment. Then, these participants could classify their commits manually allowing the comparison between our model with a human classification.

Finally, our model could be tested against Deep Learning approaches in order to further expand the results reported in this paper. For instance, we could apply both CNN (Convolutional Neural Network) and LSTM (Long Short-Term Memory) models to analyze the results reported by fastText. This is certainly an interesting path that we may take in the next step of this research as we progress with the studies about the commit classification task. We are also already working in an approach to predict software health based on the commit classification generated from this work.

Acknowledgements. This research was partially supported by Brazilian funding agencies: CAPES, CNPq (grant 424340/2016-0), and FAPEMIG (grant PPM-00651-17).

References

1. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. TACL **5**, 135–146 (2017)
2. Casalnuovo, C., Suchak, Y., Ray, B., Rubio-González, C.: Gitcproc: A tool for processing and classifying github commits. In: Proc. of the 26th ACM SIGSOFT Int'l Symposium on Software Testing and Analysis (ISSTA). pp. 396–399 (2017)
3. Crandall, D.J., Backstrom, L., Huttenlocher, D., Kleinberg, J.: Mapping the world's photos. In: Proceedings of the 18th International Conference on World Wide Web (WWW). pp. 761–770 (2009)
4. Davis, J., Goadrich, M.: The relationship between precision-recall and roc curves. In: Proceedings of the 23rd International Conference on Machine Learning. pp. 233–240. ICML '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1143844.1143874
5. Fluri, B., Wuersch, M., PInzger, M., Gall, H.: Change distilling:tree differencing for fine-grained source code change extraction. IEEE Transactions on Software Engineering **33**(11), 725–743 (Nov 2007). https://doi.org/10.1109/TSE.2007.70731
6. Golder, S.A., Huberman, B.A.: Usage patterns of collaborative tagging systems. J. Inf. Sci. **32**(2), 198–208 (Apr 2006). https://doi.org/10.1177/0165551506062337
7. Hattori, L.P., Lanza, M.: On the nature of commits. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering. pp. III–63–III–71. ASE'08, IEEE Press, Piscataway, NJ, USA (2008). https://doi.org/10.1109/ASEW.2008.4686322
8. Hindle, A., German, D.M., Godfrey, M.W., Holt, R.C.: Automatic classification of large changes into maintenance categories. In: 2009 IEEE 17th International Conference on Program Comprehension. pp. 30–39 (May 2009)
9. Hindle, A., German, D.M., Holt, R.: What do large commits tell us?: A taxonomical study of large commits. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories. pp. 99–108. MSR '08, ACM, New York, NY, USA (2008)
10. Izquierdo, J.L.C., Cosentino, V., Rolandi, B., Bergel, A., Cabot, J.: Gila: Github label analyzer. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). pp. 479–483 (March 2015)

11. Jiang, S., McMillan, C.: Towards automatic generation of short summaries of commits. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). pp. 320–323 (May 2017). https://doi.org/10.1109/ICPC.2017.12
12. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2. pp. 1137–1143. IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995)
13. Levin, S., Yehudai, A.: Using temporal and semantic developer-level information to predict maintenance activity profiles. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 463–467 (Oct 2016). https://doi.org/10.1109/ICSME.2016.21
14. Levin, S., Yehudai, A.: 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. pp. 97–106. PROMISE, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3127005.3127016
15. Lientz, B.P., Swanson, E.B., Tompkins, G.E.: Characteristics of application software maintenance. Commun. ACM **21**(6), 466–471 (Jun 1978). https://doi.org/10.1145/359511.359522
16. Mauczka, A., Brosch, F., Schanes, C., Grechenig, T.: Dataset of developer-labeled commit messages. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. pp. 490–493 (May 2015). https://doi.org/10.1109/MSR.2015.71
17. Mockus, A., Votta, L.G.: Identifying reasons for software changes using historic databases. In: Proceedings of the International Conference on Software Maintenance (ICSM'00). pp. 120–. ICSM '00, IEEE Computer Society, Washington, DC, USA (2000)
18. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (Nov 2011)
19. Ramos, J.: Using tf-idf to determine word relevance in document queries (1999)
20. Rosen, C., Grawi, B., Shihab, E.: Commit guru: Analytics and risk prediction of software commits. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. pp. 966–969. ESEC/FSE 2015, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2786805.2803183
21. Safdari, N.: Project title. https://github.com/nxs5899/Multi-Class-Text-Classification—-Random-Forest (2018)
22. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: Proceedings of the 2005 International Workshop on Mining Software Repositories. pp. 1–5. MSR '05, ACM, New York, NY, USA (2005). https://doi.org/10.1145/1082983.1083147
23. Viggiato, M., Oliveira, J., Figueiredo, E., Jamshidi, P., Kastner, C.: How do code changes evolve in different platforms? a mining-based investigation. In: 35th IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 218–222 (2019)
24. Wu, H.C., Luk, R.W.P., Wong, K.F., Kwok, K.L.: Interpreting tf-idf term weights as making relevance decisions. ACM Trans. Inf. Syst. **26**(3), 13:1–13:37 (Jun 2008). https://doi.org/10.1145/1361684.1361686