PROCEEDINGS OF SCIENCE

# Using Natural Language Processing to Extract Information from Unstructured Code-Change Version Control Data: Lessons Learned

**Elisabetta Ronchieri,**[a,*] **Marco Canaparo**[a] **and Yue Yang**[b]

[a] *INFN-CNAF,*
*Viale Berti Pichat 6/2, Bologna, Italy*
[b] *Department of Statistical Sciences, University of Bologna*
*Via delle Belle Arti, 41, Bologna, Italy*
*E-mail:* elisabetta.ronchieri@cnaf.infn.it, marco.canaparo@cnaf.infn.it, yue.yang3@studio.unibo.it

Context: Natural Language Processing (NLP) is a branch of artificial intelligence that extracts information from language. In the field of software engineering, NLP has been employed to extract key information from free-form text, to generate models from the analysis of text or to categorize code changes according to their commit messages. In literature, most of the approaches NLP-based focused on the impact of code changes on program execution or software architecture.
Objective: In this study, we have applied NLP to code-change data to identify patterns of software code modifications and used Machine Learning techniques to build a model that determines how software has evolved over time and identifies area of code that presents problems.
Method: Considering that software projects use version control systems, such as github, to manage their code, we have collected software information by using git commands. These data contain different unstructured information about the various files in a project. Each modification entry includes a message that explains the reasons for the change. According to the content of the message, it is possible to identify key terms that can be used during the classification of the entries.
Results: In this study, we have considered the change history of software available on github to the High Energy Physics community. With the use of NLP techniques we have cleaned the messages and extracted some key terms to categorize both software problems and some other changes performed by developers, like the addition of a third party dependency or a script that starts a given service. We have built a code change dictionary combining the terms already in existing literature with the ones gathered directly from the software and its github repository. Finally, we have applied some Machine Learning (ML) techniques to determine any connection between code changes and software problems: we have removed redundant entries to avoid any bias in the outcomes of the ML techniques.
Conclusion: We show in detail our approach adopted to construct historical code change datasets of categorized commit messages by following a multi-label classification methodology. Our model performance seems promising in terms of accuracy, precision, recall and F1-score.

*International Symposium on Grids & Clouds 2021, ISGC2021 22-26 March 2021*
*Academia Sinica, Taipei, Taiwan (online)*

*Speaker





PoS(ISGC2021)025

## 1. Introduction

Natural Language Processing (NLP) is an area of artificial intelligence that uses computer science to elaborate on human languages. NLP has been applied to many fields, such as text classification and clustering, information retrieval, filtering and extraction, voice recognition, email categorization and machine translation [1, 2]. Many NLP techniques have been employed to extract key information from free-form text or to generate models from the analysis of text. They have also been applied to categorize code changes according to their commit messages.

NLP can be applied to every phase of the Software Development Life Cycle (SDLC) that deals with software development through the usage of tools, methods, processes and techniques [3]. NLP has been proved to be useful when the artifacts of SDLC phase or activity are plain text [4]. NLP has been applied to Software Engineering (SE) tasks in order conduct different activities, such as the detection of developers' emotions [5], the detection of the opinions of a software product [6], the improvement of test case selection [7], code review [8], requirements engineering [9] and the detection of error-prone software components [10].

Some NLP-based approaches have focused on code changes and their consequences in terms of code review and mapping of bug reports to relevant files [11, 12]. Software systems are in continuous evolution [13], their changes may be imposed by some modifications in requirements, by the introduction of new features, by the resolution of bugs or by code refactoring; furthermore, changes may occur both during the development and maintenance of a software systems. These modifications contribute to increasing the complexity of a software system and can be the cause of the introduction of new defects.

All the data produced over time by software projects are stored in different kinds of software repositories. Version Control and Source Code Management System (SCM) repositories contain all the changes to which the different source code files are subjected [14]. Github is the world's largest version control system and SCM system with more than 100 million publicly available repositories [12]. Code constitutes only a small part of software repositories that usually include issue descriptions, history of code changes with their comments, various logs an other unstructured data. Mining Software Repositories (MSR) is the data-mining based activity that can provide new insights about how software has been developed and maintained, and consequently limit its defect proneness, improve maintainability reducing its change-proneness [15].

Change prediction is the field of software engineering responsible for identifying the software modules more likely to be modified in the future, this help developers both for maintenance operations and to monitor the complexity of source code [16]. Change prediction models represent the most common way to determine change-prone software modules [17]. In this context, Machine Learning (ML) techniques have been largely exploited to relate a set of independent variables extracted from code repositories to a dependent variable i.e. the change-proneness of a module [18]. ML approaches employed in previous literature show their general effectiveness, however, it has been noticed that they produce different results according to the different ML techniques [16]. In our work, we have used ML techniques both for clustering and classification activities.

Commit messages represent the natural description of software changes: developers usually upload code to a version control system together with a short commit message that explains the purpose of the change. Commit messages are important because they help to understand the reason

behind a software change, contributing to monitoring the evolution of a software project [19].

However, time may affect negatively many types of documentation, including commit messages [20, 21]. These issues lead to commit messages that are insufficient, inconsistent with the projects updates or they may include informal languages. In addition, one commit message may include more than one update [5]. Many previous approaches in literature assumed that one commit message could be associated with only one label, on the contrary, developers often perform more tasks in a single commit [12].

In this study, we aim at building a historical code change dataset composed of commit messages classified according to their types of changes and corrections. On the dataset, systematic queries can be executed to show the evolution of a software project. In addition, we will follow a multi-label classification approach, i.e. a commit message can be associated with more than one class label.

Our final goal is to provide developers - in the High Energy Physics (HEP) context - with a tool for the automatic classification of commit messages. We believe our tool would be helpful to monitor development and maintenance activities at any point in time and to improve software development practice.

In order to highlight the main parts of our research we would like to answer the following research questions:

RQ1: What has been the impact on NLP techniques for the construction of our dictionary and the feature definition?

RQ2: Is the constructed dataset suitable for code-change prediction?

RQ3: Which ML algorithms or algorithm family perform the best in code-change prediction?

In the following part of the paper, we start by describing previous literature on similar studies in section 2. We detail our approach in section 3. In section 4 we display our results and finally in section 5 we draw our conclusions.

## 2. Related work

In previous literature, there have been other studies that have tackled the issue of categorizing commit messages into their respective categories with the aim of identifying existing key words or, as in our research, applying ML techniques on their features.

Mondal et al. [22] explored the relation between free-form natural language text (e.g. developers' communication and commit messages) and automated architectural change analysis. The authors' approach is based on different kinds of natural language text and employs the tf-idf and term-graphs techniques to detect change related samples, achieving satisfactory results (i.e., 61% F1-score). As regards the classification of architectural changes their results are limited.

Sarwar et al. [12] proposed a transformer-based neural network to address multi-label commit message classification problem. The obtained model achievied a F1-score of 87% and out-performed previous commit message classification approaches.

Levin et al. [23] introduced a novel method for the automatic classification of commit messages into maintenance activities by using source code changes. The method reached a promising accuracy result of 76% and Cohen's kappa of 63% for the test dataset.

Gharbi et al. [24] presented a multi-label active learning-based approach to address the problem of the classification of commit messages. The authors' results showed a good performance in terms of F1-score with an average of 45.79%.

Mauczka et al. [25] aimed at surveying open source project contributors to classify their changes through three different classification methods. The authors managed to build a dataset of 967 classified commits each enriched with meta-information on functional requirements if present.

Hindle et al. [26] focused on the automatic classification of large changes into various categories of maintenance tasks using ML techniques. The authors started by manually classifying 1% of commit messages and they employed them as training set of a ML classifier. The achieved accuracy was around 35-70% in a single project, 52% in cross-project classification.

Fischer et al. [27] introduced a dataset composed of version and bug report data. The authors showed some query examples about the software evolution analysis.

To the best of our knowledge, our study presents a novel approach based on a multi-label classification methodology that enabled us to construct a historical change dataset composed of commit messages; furthermore, we have applied these techniques on High Energy Physics software projects. As shown in section 4 our outcomes are promising since the measured values of accuracy metric is 95.8% and of F1-score is 95.6%.

## 3. Approach

In this section, we outline the followed approach to extract information from code-change version control data. First, we briefly introduce the overall framework, and later we describe the various steps that characterize our approach.

Figure 1 shows the workflow we have followed to perform our work. It contains six steps, including: dataset construction based on information available on github, data preprocessing to clean messages and remove unuseful terms, feature extraction, clustering with unsupervised techniques, classification, model training to predict code change proneness and performance evaluation. The first three steps are those that require experts revision and where we have dedicated almost 90% of our time.

### 3.1 Dataset construction

We first identified a set of software projects from the high energy physics domain (see Table 1) available on the github repository [15]. So far we have selected 167 projects. Table 1 shows the projects involved in our study: from ALISW [1] we have extracted 102 projects, from LHCB [2] 107 projects, from CMS-SW [3], 41 projects, from ROOT [4] and GEANT4 [5] 1 project.

To construct a set of datasets, we have collected all the commit messages available up to November 2020 by using `git log` command. We have performed some data cleaning operations by using python-based framework and R library and gathered some messages statistics per project

[1] https://github.com/alisw
[2] https://github.com/lhcb
[3] https://github.com/cms-sw
[4] https://github.com/root-project/root
[5] https://github.com/Geant4/geant4

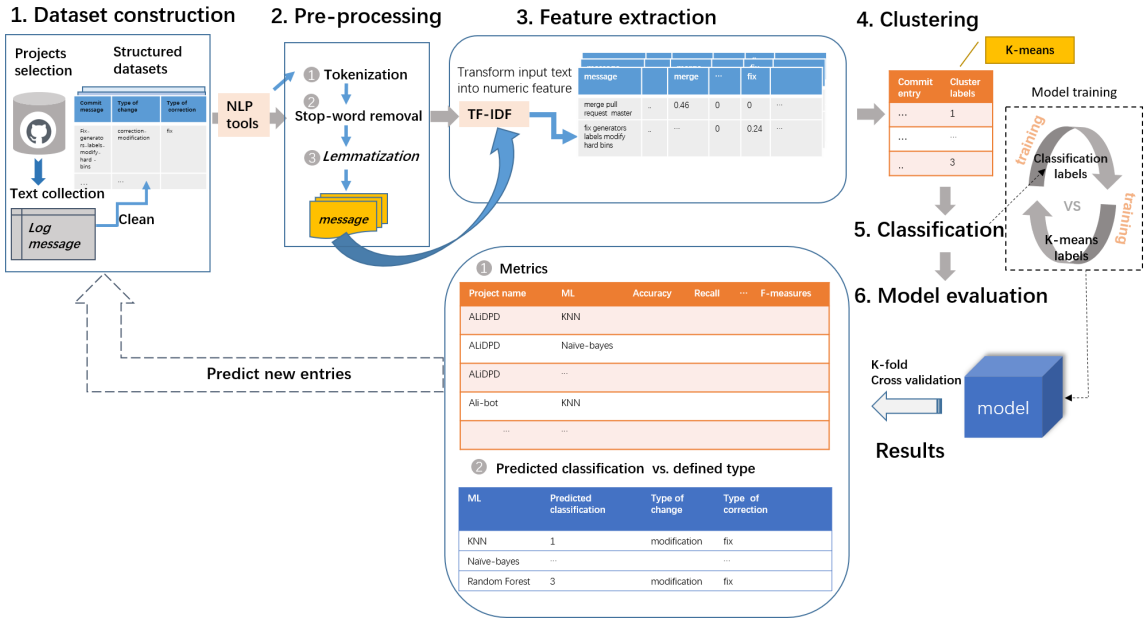


**Figure 1:** Approach workflow.

| ALISW | LHCB | CMS-SW | ROOT | GEANT4 |
|---|---|---|---|---|
| AliDPG | starterkit-lessons | cms-prs | root | geant4 |
| AliRoot | starterkit-ci | cmssw | | |
| alidist | glossary | cms-sw.github.io | | |
| AliPhysics | analysis-essentials | cmssw-config | | |
| AliGenerators | starterkit | cmssw-cfipython | | |
| ali-bot | developkit-lessons | cms-bot | | |
| docks | ostap-tutorials | cms-docker | | |
| dim | DevelopKit | genproductions | | |
| delphes | gitbook-plugin-panels | SCRAM | | |
| alibuild | starterkit-receipts | root | | |
| pythia6 | bender-tutorials | cmssdt-web | | |
| homebrew-system-deps | Condorcet | cmssdt-wiki | | |
| create-pull-request | second-analysis-steps | cmssw-wm-tools | | |
| arrow | first-analysis-steps | cmssdt-ib | | |
| RooUnfold | opendata-project | pkgtools | | |
| alisw.github.io | plugin-codesnippet | cmspkg | | |

**Table 1:** Software Projects on GitHub.

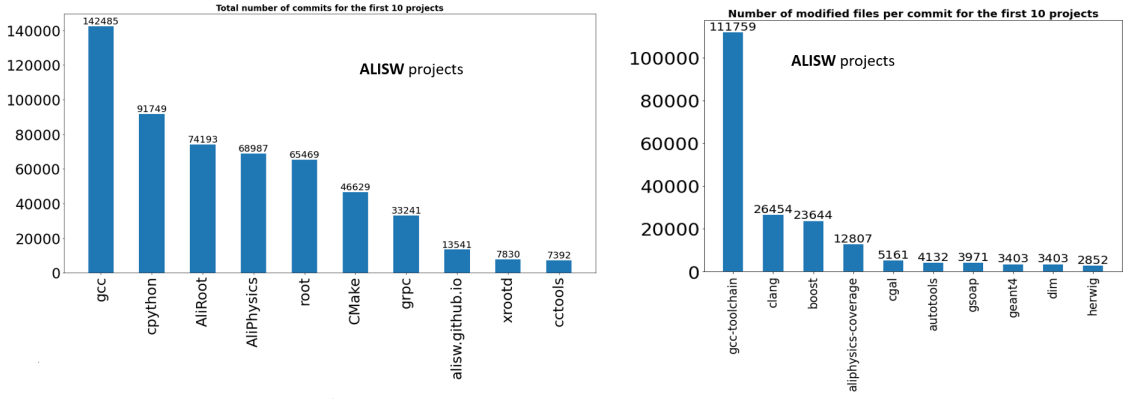


**Figure 2:** Commit Messages Statistics.

and projects groups. In particular, we have considered: the total number of days which the messages refer to, the number of authors involved in the operations, the total number of commits performed by developers, the names of the modified files, the number of modified files, the number of authors per commit; the range of project time period [start date, end date]. Figure 2 shows two different statistics for the first 10 ALISW projects on github: on the left side there is the total number of commits per project, on the right side there is the number of modified files per commit and per project.

During these activities, we have also identified categories for each commit message according to its type of change. We have extracted key terms from the various commit messages stored in the `git log` files, introducing further categories with respect to existing taxonomy [28] that we have considered too limiting according to the content of each file. Therefore, categories of changes have also chosen in order to cover corrective activities performed by software developers. Figure 3 shows some of the identified categories that we have used in the dataset (the terms in bold derive from Hindle et al. [28]), such as *installation* that refers to changes that were required to improve software installation, and *cancellation* that refers to changes that were required e.g. to remove piece of code, files, piece of documentation. Figure 3 highlights commit messages' key terms that belong to the corrective category: *crash* considers changes that were performed to avoid a software application to stop functioning correctly and exit; *patch* is a set of changes that was performed to update, fix or improve part of a software. Some other commit messages' key terms of the other categories are: cancel, remove, drop and delete for the *cancellation* category; improvement, report error, hardcode, clean and move for the *improvement* category.

### 3.2 Data preprocessing

The data preprocessing has implied the usage of NLP tools to clean the initial message and identify potential features. Figure 4 shows the steps we have conducted: the tokenization activity splits the message into words that have been spelled correctly; the filtering operation removes stop words, punctuation, numbers, non-English words; the tagging phase keeps nouns, verbs and adjectives; and the lemmatization process of such messages reduces the size of features' matrix. To reduce the size of the feature matrix we have applied the lemmatization method [29, 30] that is the process of determining the lemma, i.e. the "dictionary form", of a given word where different

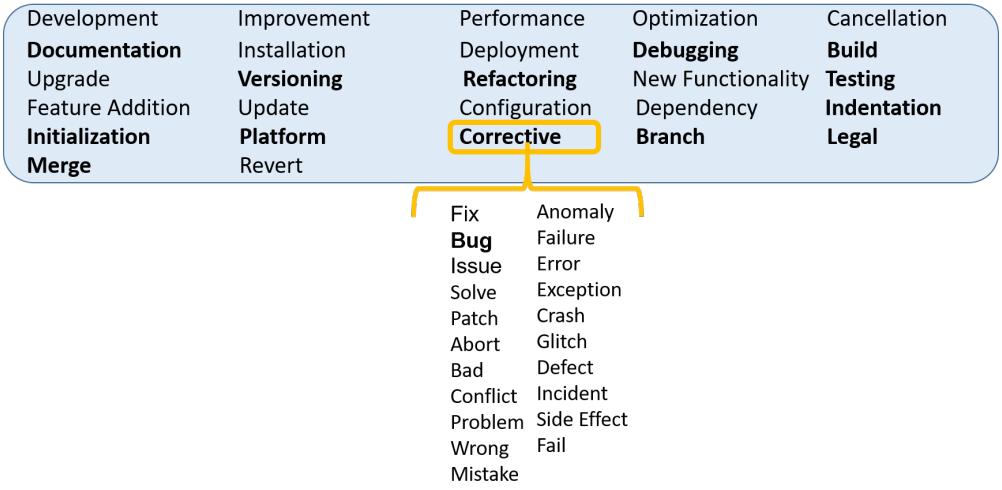


**Figure 3:** Categories of Changes: commit messages' key terms for the corrective category.

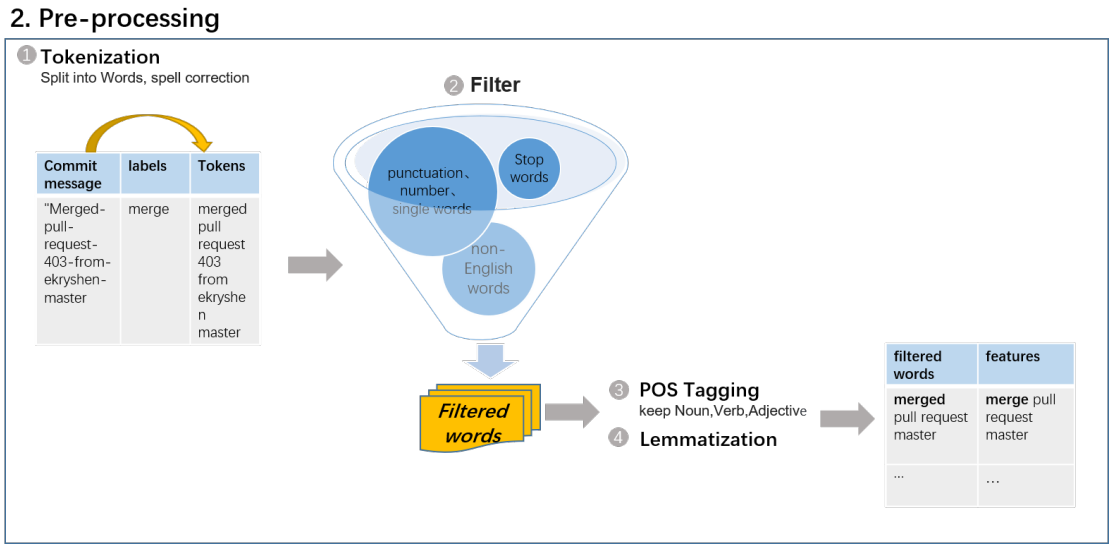


**Figure 4:** Preprocessing Phase.

inflected forms of a word are possible. For instance, the two forms of the word "added" and "adding" can be reduced to the lemma "add".

During this phase we have interactively performed a manual check of the results obtained to avoid removing useful terms.

### 3.3 Feature Extraction

At this stage we have first vectorized the features, putting 1 when the word is in the message, 0 otherwise. Secondly, we have weighted the various terms by employing the TF-IDF technique [31, 32] to trasform each input text into a numeric feature. TF-IDF is an information retrieval technique used to evaluate how relevant a word (i.e. term) is to a document (commit message) in a collection of

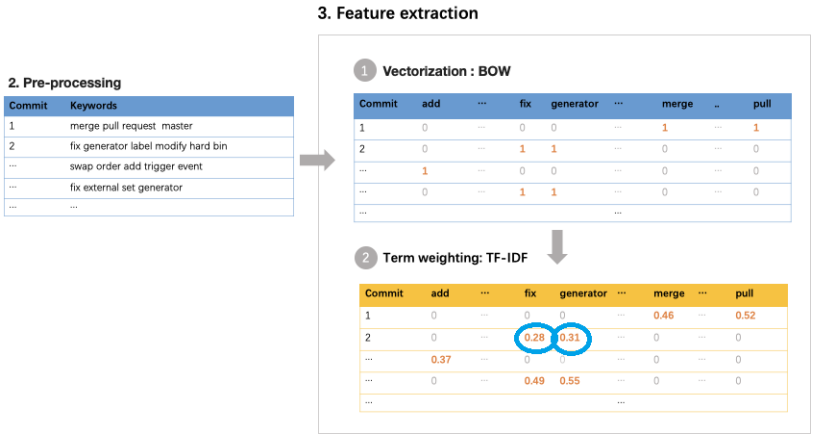


**Figure 5:** Feature Extraction with the TF-IDF technique.

documents (log messages): the document is the commit message, while the collection of documents is the log messages. Figure 5 details how the process of feature extraction works.

The calculation of TF-IDF = tf($t$, $d$) × idf($t$) uses two metrics:

TF tf($t$, $d$) is the term frequency (i.e. the number of times) of the term $t$ in a document $d$;

IDF idf($t$) = $log(\frac{N}{1+|d \in D : t \in d|})$ is the inverse document frequency of the term $t$ across a set of documents $D$: The number 1 is used when the term $t$ is not in the corpus; $|\{d \in D : t \in d\}|$ is the number of documents where the term $t$ appears and tf($t$, $d$) $\neq 0$, $D$ is the set of documents $d$; $N$ is the number of documents in the corpus $|D|$.

The higher the TF-IDF score, the more relevant that term is in that particular document. The two blue cicles in Figure 5 highlight details about feature extraction phase. The row values in the table show the TF-IDF score for the terms fix and generator: these two terms receive a low document frequency in the whole collection of log messages.

### 3.4 Clustering

Based on the result obtained during the feature extraction, we have applied the $k$-means clustering technique on feature matrix to identify clusters of commit messages and group the commit messages into their categories. To determine the optimal number of clusters $k$, we have applied the elbow method and silhouette score whose representation is shown in Figure 6 on the top and bottom sides respectively.

The plots on the top side show the $k$ values based on the elbow method for the AliDPD and alidist projects. The circles in Figure 6 highlight likely $k$ values: the selected value is obtained with the elbow method whose result is shown by the vertical dot line. The reported score computes the sum of the squared distances from each point to the assigned center.

The plots on the bottom side show the $k$ values based on the silhouette score for the AliDPD and alidis projects. The selected value is obtained with the silhouette method whose result is shown by the vertical dot line. The reported score considers the maximum curvature: in this case the score is calculated by using the mean intra-cluster distance and the mean near-cluster distance.

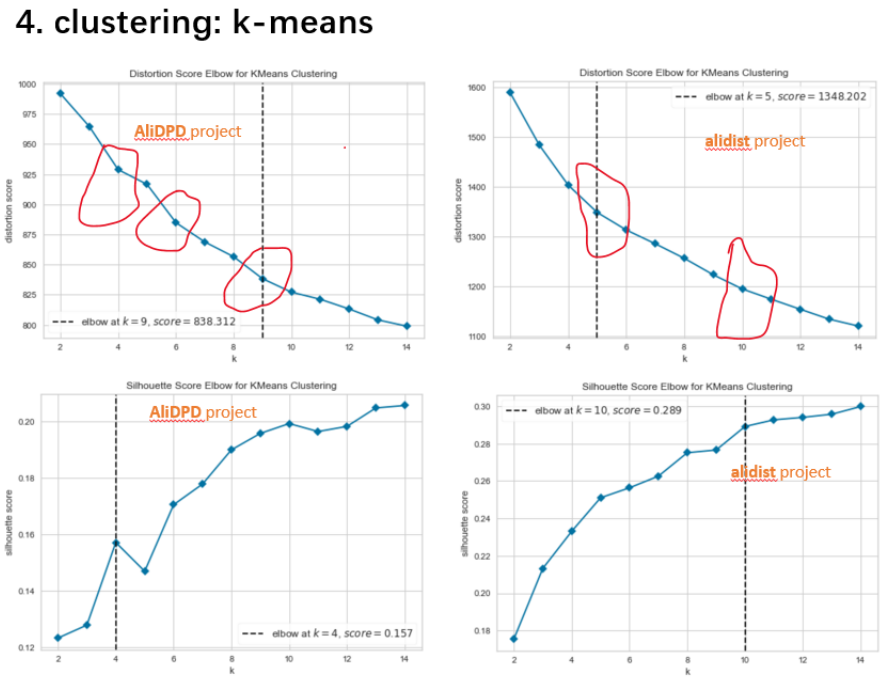


**Figure 6:** Computing the number of cluster $k$ with the elbow method and the silhouette method.

### 3.5 Model Evaluation

This phase involves the use of ML-based classification techniques on our training datasets. The data obtained at the clustering step is split in training and testing datasets in order to build a model that is able to identify how the different code-changes clusters together.

The evaluation of ML techniques' has been measured by performance metrics' values (such as F1-score, precision and recall) as well as by the comparison between the classification obtained with our model and the manually derived category. Our models were then employed to categorize other entries (i.e. other commit messages). We have used the following ML algorithms: NaiveBayes, NaiveBayesMultinomial, RandomForest and DecisionTree.

We have considered the whole code change datasets, composed of one dataset per project, through the application of ML techniques. We have split all the datasets in two subsets, one for the validation of our model and composed of the 30% of the entries, the other to perform the training of the model and made up of the remaining 70% of the entries. We have compared the different ML techniques through some performance metrics, such as:

- F1-score=$2 \times \frac{(precision \times recall)}{(precision + recall)}$
- precision = $\frac{tp}{tp+fp}$
- recall = $\frac{tp}{tp+fn}$
- accuracy = number of correct predictions/ total number of predictions

$tp$ is the number of true positives classified by the model, $fn$ is the number of false negatives classified by the model, $fp$ is the number of false positives classified by the model.

| Project Name | #features without NLP | #features with NLP | feature reduction percentage |
|---|---|---|---|
| 1 | 1660 | 351 | 78.9% |
| 2 | 974 | 411 | 57.8% |
| 3 | 2487 | 483 | 80.6% |
| 4 | 500 | 160 | 68.0% |
| 5 | 1382 | 321 | 76.8% |
| 6 | 405 | 177 | 56.3% |

**Table 2:** Comparison of size features with and without NLP

| ML Technique Name | Accuracy | Recall | Precision | F1-score |
|---|---|---|---|---|
| Decision Tree | 0.958 | 0.977 | 0.958 | 0.957 |
| Multinomial naive Bayes | 0.596 | 0.790 | 0.596 | 0.506 |
| Naive Bayes | 0.548 | 0.760 | 0.548 | 0.519 |
| Random Forest | 0.910 | 0.938 | 0.910 | 0.908 |

**Table 3:** Comparison of the different ML techniques - average values are reported

F1-score determines the entries (e.g. modules or files associated to commit messages) that were predicted as belonging to a particular cluster (a combination of types of changes and corrections).

## 4. Results

In this work, we have risen the following research questions.

RQ1: What has been the impact on NLP techniques for the construction of our dictionary and feature definition? NLP techniques proved to have an important impact on our work as the number of features have decreased compared to our initial study presented at IEEE NSS MIC 2020 (under publication) [33] that was conducted without NLP. NLP has contributed to remove unimportant and redundant words. Table 2 shows the different size features achieved by the two studies for six projects, the one with NLP allowed to reduce the number of features of the model.

RQ2: Is the constructed database suitable for code-change prediction? Based on the F1-score metric, our analysis shows that the dataset is a suitable input for code-change prediction models that rely on ML techniques. Table 3 shows that F1-score reaches an average value up to 0.95660 that can be considered a promising result.

In our initial study presented at IEEE NSS MIC 2020 (under publication) [33], the F1-score reached values up to 0.8210 for the considered ML techniques (also listed in Table 3). In this study the F1-score metric has improved. However, before making any conclusion we have to apply the study to the whole projects and checking the correctness of the produced datasets.

RQ3: Which ML algorithms or algorithm family perform the best in code-change prediction? Table 3 shows that Decision Tree and Random Forest have the highest average F1-score values in the majority of the examined projects.

Our research is the result of some choices we have made and that imply some threats to validity. The first choice is to have considered only HEP projects stored in GitHub, this may have led to a projects' bias; the second threat concerns the programming language: the majority of the software projects are written in C++, this may have affected the text of commit messages; the last threat is due to the lack of a general agreement on how commit messages can be labeled if the labelling process is performed by different researches. However, in order to overcome these limitations, we have relied on peer-reviewing and on a partial automation of the process.

## 5. Conclusions

We have suggested a novel approach for building a historical code change dataset of categorized commit messages by following a multi-label classification approach. Our methodology is composed of six steps: dataset construction, word preprocessing, feature extraction, clustering, model evaluation and finally results.

The aim of this paper is twofold: firstly, we have built a dataset composed of structured commit messages classified according to their types of change or correction where systematic queries can be run; secondly, we have followed a multi-label classification approach which enable to categorize one commit message to more than one class lable. In order to achieve our aims, we have created a dictionary for labelling commit messages based on categories both derived from existing literature and found in the analyzed commit messages. Our final objective is to provide developers with a tool for the automatic classification of commit messages. We believe that our work may help monitoring and improving both the development and maintenance processes of software projects.

In the future, we plan to extend our research by analysing the commit messages that have been misclassified, we will include other commit messsages that might not be available in the current datasets. Furthermore, we would like to incorporate developers' community feedback to define more suitable code change categories and finally we will aim at finilizing our tool for the automatic classification of commit messages.

## References

[1] S. Falkenstine, A. Thornton and B. Meiners, *Natural Language Processing for Autonomous Identification of Impactful Changes to Specification Documents*, in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pp. 1–9, 2020, DOI.

[2] S. Wang, F. Ren and H. Lu, *A review of the application of natural language processing in clinical medicine*, in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 2725–2730, 2018, DOI.

[3] P. Jalote, *An Integrated Approach to Software Engineering*, Texts in Computer Science. Springer, Boston, MA, 2005, 10.1007/0-387-28132-0.

[4] P. Yalla and N. Sharma, *Integrating Natural Language Processing and Software Engineering*, *International Journal of Software Engineering and Its Applications* **9** (2015) 127.

[5] A. S. M. Venigalla and S. Chimalakonda, *Understanding Emotions of Developer Community Towards Software Documentation*, 2021.

[6] D. Girardi, N. Novielli and Fucci, *Recognizing Developers' Emotions While Programming*, in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, (New York, NY, USA), pp. 666–677, Association for Computing Machinery, 2020, DOI.

[7] V. Garousi, S. Bauer and M. Felderer, *NLP-assisted software testing: A systematic mapping of the literature*, *Information and Software Technology* **126** (2020) 106321.

[8] J. Siow, C. Gao, L. Fan, S. Chen and Y. Liu, *CORE: Automating Review Recommendation for Code Changes*, in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019.

[9] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca et al., *Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study*, 2020.

[10] X. Ye, R. Bunescu and C. Liu, *Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation*, *IEEE Transactions on Software Engineering* **42** (2016) 379.

[11] F. Gilson and D. Weyns, *When Natural Language Processing Jumps into Collaborative Software Engineering*, in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 238–241, 2019, DOI.

[12] M. U. Sarwar, S. Zafar, M. W. Mkaouer, G. S. Walia and M. Z. Malik, *Multi-label Classification of Commit Messages using Transfer Learning*, in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 37–42, 2020, DOI.

[13] G. Catolino and F. Ferrucci, *Ensemble techniques for software change prediction: A preliminary investigation*, in *2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, pp. 25–30, 2018, DOI.

[14] D. Rodriguez, I. Herraiz and R. Harrison, *On software engineering repositories and their open problems*, in *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*, pp. 52–56, 2012, DOI.

[15] V. A. Luzgin and I. I. Kholod, *Overview of Mining Software Repositories*, in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp. 400–404, 2020, DOI.

[16] G. Catolino, F. Palomba, A. De Lucia, F. Ferrucci and A. Zaidman, *Enhancing change prediction models using developer-related factors*, *Journal of Systems and Software* **143** (2018) 14.

[17] Y. Zhou, H. Leung and B. Xu, *Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness*, *IEEE Transactions on Software Engineering* **35** (2009) 607.

[18] N. Pritam, M. Khari, L. Hoang Son, R. Kumar, S. Jha, I. Priyadarshini et al., *Assessment of Code Smell for Predicting Class Change Proneness Using Machine Learning*, *IEEE Access* **7** (2019) 37414.

[19] S. Jiang, A. Armaly and C. McMillan, *Automatically generating commit messages from diffs using neural machine translation*, in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 135–146, Oct, 2017, DOI.

[20] Y. Huang, N. Jia, H.-J. Zhou, X.-P. Chen, Z.-B. Zheng and M.-D. Tang, *Learning Human-Written Commit Messages to Document Code Changes*, *Journal of Computer Science and Technology* **35** (2020) 1258.

[21] L. Hattori, M. D'Ambros, M. Lanza and M. Lungu, *Software Evolution Comprehension: Replay to the Rescue*, in *2011 IEEE 19th International Conference on Program Comprehension*, pp. 161–170, 2011, DOI.

[22] A. K. Mondal, B. Roy and K. A. Schneider, *An Exploratory Study on Automatic Architectural Change Analysis Using Natural Language Processing Techniques*, in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 62–73, 2019, DOI.

[23] 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, (New York, NY, USA), pp. 97–106, Association for Computing Machinery, 2017, DOI.

[24] S. Gharbi, M. W. Mkaouer, I. Jenhani and M. B. Messaoud, *On the Classification of Software Change Messages Using Multi-Label Active Learning*, in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, (New York, NY, USA), pp. 1760–1767, Association for Computing Machinery, 2019, DOI.

[25] A. Mauczka, F. Brosch, C. Schanes and T. Grechenig, *Dataset of Developer-Labeled Commit Messages*, in *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pp. 490–493, IEEE Press, 2015.

[26] A. Hindle, D. M. German, M. W. Godfrey and R. C. Holt, *Automatic classication of large changes into maintenance categories*, in *2009 IEEE 17th International Conference on Program Comprehension*, pp. 30–39, 2009, DOI.

[27] M. Fischer, M. Pinzger and H. Gall, *Populating a Release History Database from version control and bug tracking systems*, in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, pp. 23–32, 2003, DOI.

[28] A. Hindle, D. M. German and R. Holt, *What Do Large Commits Tell Us? A Taxonomical Study of Large Commits*, in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, (New York, NY, USA), p. 99–108, Association for Computing Machinery, 2008, DOI.

[29] S. Khatiwada, M. Kelly and A. Mahmoud, *STAC: A tool for Static Textual Analysis of Code*, in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pp. 1–3, 2016, DOI.

[30] O. Ozturkmenoglu and A. Alpkocak, *Comparison of different lemmatization approaches for information retrieval on Turkish text collection*, in *2012 International Symposium on Innovations in Intelligent Systems and Applications*, pp. 1–5, 2012, DOI.

[31] Z. Jiang, B. Gao, Y. He, Y. Han, P. Doyle and Q. Zhu, *Text Classification Using Novel Term Weighting Scheme-Based Improved TF-IDF for Internet Media Reports*, *Mathematical Problems in Engineering* **2021** (2021) .

[32] L. H. Patil and M. Atique, *A novel approach for feature selection method TF-IDF in document clustering*, in *2013 3rd IEEE International Advance Computing Conference (IACC)*, pp. 858–862, 2013, DOI.

[33] E. Ronchieri, Y. Yang, M. Canaparo, A. Costantini, D. C. Duma and D. Salomoni, *A new code change prediction dataset: a case study based on HEP software*, in *Under Publication at IEEE NSS MIC 2020*, 2020.