 doi:10.20944/preprints202209.0309.v1

Article

Natural Language Processing Application on Commit Messages: a case study on HEP software

Yue Yang [1], Elisabetta Ronchieri [1,2,*], Marco Canaparo [2,*]

[1] Department of Statistical Sciences, University of Bologna, Italy
[2] INFN CNAF, Bologna, Italy
* Correspondence: elisabetta.ronchieri@cnaf.infn.it (E.R.)
Tel.: +39-0512095072

Abstract:

Background: Version Control and Source Code Management Systems, such as GitHub, contain large amount of unstructured historical information of software projects. Recent studies have introduced Natural Language Processing (NLP) to help software engineers retrieve information from very large collection of unstructured data. In this study, we have extended our previous study by increasing our datasets and ML and clustering techniques.

Method: We have followed a complex methodology made up of various steps. Starting from the raw commit messages we have employed NLP techniques to build a structured database. We have extracted their main features and used as input of different clustering algorithms. Once labelled each entry, we have applied supervised machine learning techniques to build a prediction and classification model.

Results: We have developed a machine learning-based model to automatically classify commit messages of a software project. Our model exploits a ground-truth dataset which includes commit messages obtained from various GitHub projects belonging to the HEP context.

Conclusions: The contribution of this paper is two-fold: it proposes a ground-truth database; it provides a machine learning prediction model. They automatically identify the more change-proneness areas of code. Our model has obtained a very high average precision, recall and F1-score.

Keywords: machine learning; natural language processing; commit messages; change prediction model

1. Introduction

In the last decade, software development from a collaborative activity has turned into a social phenomenon which relies on social coding platforms such as GitHub, Bitbucket and Gitlab [1,2]. These platforms provide users with integrating mechanisms such as issue reporting, pull requests, commenting and reviewing support .

During software development, all the produced data are stored in software repositories. Version Control and Source Management System (SCMS) repositories include all the changes of all the open source software projects. GitHub is the largest SCMS in the word [3] and contains a wealth of data for each open source project e.g., code, issue description and code changes. Furthermore, GitHub supports the commit operation and keeps trace of all changes produced by developers to a file or a set of files (containing either code or text).

Each commit is linked to various types of other data, such as the changed files, a description of changes (i.e., commit message), and the author of the operation. A commit message is an unstructured text [4] that is generated when a programmer applies a change to the GitHub project, and includes a short description of what has been done. Commit messages are widely recommended, because they help developers to trace the rationale behind a change and software evolution. The changes can be considered as a record of functionality additions and bug repairs [5].

 doi:10.20944/preprints202209.0309.v1

Unstructured data in software repositories have grown exponentially, as a consequence the mining of unstructured data (MUD) [4] has become a popular research area in the software engineering community. Some applications of MUD techniques are the automatic generation of documentation, code change analysis, and bug proneness prediction.

The GitHub service provides a log operation that retrieves all the information linked to commits. Through log documentation one can monitor the evaluation of software projects and understand the development and maintenance activities. However, code change analysis through commit messages is a challenging task because the text is usually inconsistent (e.g. it can contain change for problem fix and new development) and composed of informal language (e.g. the name of a package). These are the main barriers when it is necessary to label the commit messages. Natural Language Processing (NLP) has been applied to categorize code changes according to their commit messages.

During the Software Development process, NLP can be a useful tool provided that all the artifacts produced are plain text [6,7]. More in detail, Software Engineering exploits NLP to conduct a variety of activities that range from the detection of developers' emotion [8] and opinions of a software product [8] to the improvement of test case selection [9], code review [10], requirements engineering [11] and the detection of error-prone software components [12]

Over time, NLP has been applied on code changes and their consequences in terms of code review and mapping of bug reports to relevant files [3,13]. Changes in code can be caused by several factors such as some modifications in requirements, the introduction of new features, the resolution of bugs or code refactoring; and they may occur both during the development and maintenance of a software systems. Changes increase the complexity of a software system and can be the cause of the introduction of new defects. [14]

The identification of change-prone software modules is the subject of code change prediction activity. In the software engineering context, change prediction contributes to help in maintenance operations and in monitoring code source complexity [15]. Change-prone software modules are identified through the application of change prediction models [16] that are usually Machine Learning-based. Many Machine Learning (ML) techniques have been exploited to relate code features to change proneness of a software module [17] and they have generally proven their effectiveness; nevertheless, their results seem to be affected by the used ML technique [15]. In our work, we have used ML techniques both for clustering and classification activities.

In this study, we extend the building of the historical code change dataset composed of commit messages classified according to their types of changes and corrections. Systematic queries can be executed to show the evolution of a software project. We consider a multi-label classification approach, i.e. a commit message can be associated with more than one class label. More in detail, we have added other ML techniques, extended commit messages datasets and better described the use of clustering techniques.

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 of NLP techniques on the data preparation step?

RQ2: Which ML classification techniques have the best performance of new commits classification?

RQ3: Do the models behave differently for different projects?

In the following part of the paper, section 2 describes previous work in existing literature on similar studies. We detail our approach from section 3 to section 8. In section 9 we display our results followed by our validity analysis and finally in section 11 we draw our conclusions.

2. Related Work

This section describes some previous work related to the different topics of interest for our study.

 doi:10.20944/preprints202209.0309.v1

2.1. NLP techniques

The main steps of NLP are tokenization, part-of-speech (POS) tagging, lemmatization, stemming, stop-words removal.

1. *Tokenization* [18] splits text into single words. Usually, word tokens are separated by spaces and sentence tokens are separated by stop characters. In this study, we have applied word tokenization removing punctuation and capital letters.
2. *POS tagging* involves adding a part-of-speech category to each token in the text. Categories include nouns, verbs, adverbs, adjectives, pronouns, conjunctions and their subcategories [19]. POS tagging helps to identify the syntactic relationships between words to understand the meaning of sentences.
3. *Stemming* is used to convert words to their stems [20]. In many languages, the various syntactic forms of words are used and explained using the same basic concepts. Many words in English can be simplified into their basic forms or stems. For example, “improvement", “improved", “improving" all belong to the stem “improve".
4. *Lemmatization* takes into consideration the morphological analysis of the words, converting the given word to its base form in a dictionary [21]. Generally speaking, stemming recognizes the common root form of words by deleting or replacing word suffixes, while lemmatization recognizes the inflectional form of the word and returns its basic form (for example, “are" is reduced to “be").
5. *Stop-words removal* exploits words, such as articles, pronouns, and prepositions, that bring little contextual information, but their occurrence frequency is high [18]. This activity reduces the number of features and the noise in text, and help to obtain key information [20].

2.2. Clustering techniques

Clustering analysis is a type of unsupervised learning method and it aims at assigning a set of objects to different groups. The objects in the same group are more similar to each other [22].

There are some typical clustering models [22] [23] [24]: connectivity models (e.g., hierarchical clustering), centroid models (e.g., k-means clustering and k-medoid clustering), distribution models (e.g., multivariate normal distributions), density models (e.g., DBSCAN and OPTICS), subspace models (e.g., Biclustering also known as Co-clustering or two-mode-clustering), graph-based models (e.g., HCS clustering algorithm).

2.3. Classification techniques

In ML and statistics areas, classification is a type of supervised learning method that aims at identifying the categories of instances. It is based on a training dataset that contains instances and their labels (e.g., outcomes or categories)

that contains observations (or instances) whose category membership is well known.

Naive Bayes and Multinomial Naive Bayes techniques belong to the same model family. The various Naive Bayes classifiers have a different assumption about the likelihood of the features [25]. For example,

- in the Gaussian Naive Bayes classifier, the likelihood of the features is assumed to be Gaussian.
- in the Multinomial Naive Bayes classifier (that is usually used when data contains discrete features), the assumption is that features are generated from a multinomial distribution.

Logistic Regression is a special generalized linear model, which is used to explain the relationship between a dependent variable and one or more independent variables. Logistic regression can be binomial or multinomial: binary logistical regression is used for a dependent variable that has two types of outcome, and multinomial logistic regression is used for a dependent variable that has more than two types of outcome [22].

 doi:10.20944/preprints202209.0309.v1

Decision Tree is widely used in data mining, and there are two types of tree models i.e., classification trees and regression trees. They aim at predicting the value of a target variable based on input variables [22].

Bagging, also called bootstrap aggregating, is an ensemble meta-algorithm that aims at improving the model stability. Bagging trains each model using a subset that is chosen randomly from the training set. It takes the majority vote class or takes the mean value of the individual model as the final prediction result. It can be applied to some base classifiers (e.g, decision trees, K-nearest neighbor, and logistic regression).

Random Forest is a type of Bagging method, and its base classifier is a decision tree. It is an ensemble learning method mainly for classification and regression tasks. It constructs a great number of decision trees at the training phase, and its final result is the mode of the classes or the mean value of individual trees. During the training phase, it also introduces randomness for feature selection. Only a subset of features and samples are considered for each decision tree. Random Forest reduces the risk of over-fitting. AdaBoost is an adaptive boosting algorithm. The method increases the weight of the misclassified samples of the previous classifiers so that the new classifier can focus on the training samples that are prone to misclassification.

2.4. Classification of commits

Commit messages help to keep track of the changes of software on GitHub during the development and maintenance process. The true labels of these messages are the foundation of ML classification techniques. Accurate classification of commits can help to acquire knowledge about the software evolution process and detect software defect proneness. In literature there are few studies about classification of commit messages [3] [26] [27] [28]. Research on classification of commit messages is still ongoing.

Barnett et al. [29] considered the length of commit messages and the probability of a defective commit as additional explanatory variables in (just-in-time) JIT models, which aim at predicting the commits that have the probability to introduce defects in the future. Traditionally, JIT defect prediction models only consider metrics of code-change itself, including the size of change and the authors' prior experience. This study used Naive Bayes classifiers to predict the probability of becoming a defective commit. The explanatory power of the JIT models demonstrated a significant improvement for 43%-80% of studied project, and it reached 72%.

Levin et al. [30] introduced developers' information and added novel metrics that capture temporal and semantic developer-level information. They used a generalized regression modeling (GLM) in the R statistical environment to explore their dataset and build predictive models. They defined predictive models by combining traditional project-level metrics with temporal and semantic information about developer to predict code-change types (i.e., adaptive, corrective and perfective) in maintenance activity that had been defined by Mockus et al. [**?**]. Their study reached a promising predictive power with R^2 values of 0.75, 0.83, and 0.64.

Levin et al. [27] extended their research, and utilized source code changes to classify commit messages to understand maintenance activities. They created a manually labelled commit dataset and assigned each entry into three categories (i.e. adaptive, corrective and perfective) according to keywords. They used three types of datasets: keywords data, source code-change types of data, and combined data (keywords and source code-change types), and then applied classification algorithms (i.e., Random Forest, J48, and Gradient Boosting Machine). Their study finally reached a promising accuracy of 76% and Cohen's kappa of 63%.

Gharbi et al. [28] used the TF-IDF method to extract useful features (i.e. keywords) from commit messages text, and applied active learning to reduce the workload of labelling commit messages. They used multi-label active learning with auxiliary learner strategy and logistic regression model classifier to set up an iteration mechanism, and the classifier model was trained on the updated training set and tested on a separate testing set. They

 doi:10.20944/preprints202209.0309.v1

achieved the best performance concerning hamming loss with an average of 0.05 and a good performance for F1 score with an average of 45.79%.

Sarwar et al. [3] proposed an off-the-shelf neural network, called DistilBERT, and fine-tuned it for the commit message classification problem. They considered code-change classifications, e.g., bug fix, feature addition and performance improvement. These modifications contributed to increasing the complexity of a software system, causing the risk of new defects. They utilized an NLTK package to preprocessing data and mutually labelled commit messages with multi-label. Their model reached an F1-score of 87%.

2.5. Clustering analysis of commits

The above studies considered labelled commits for classification. In reality, labeling commits have many problems. Commit submissions are usually huge, so manual labeling involves a lot of work. Second, manual labeling can easily produce bias. Different annotators may assign different labels for the same commit message, influenced by their subjective judgment or individuals' experience. Thirdly, without uniform text structure and formal language for commit messages, the classification of commit messages is a challenging task. Some researchers have explored commits information with unsupervised machine learning techniques.

Zhong et al. [31] used unsupervised learning for expert-based software quality estimation. They firstly clustered a great number of software modules into some groups and invited qualified experts to label each cluster as fault-prone or not fault-prone. Clustering techniques performed in this study were k-means and neutral gas, and results demonstrated that the netural-gas algorithm outperformed k-means concerning MES and average purity.

P. Hattori et al. [32] classified commits into four categories. Unlike categories proposed by Swanson's classification of maintenance activities, they proposed classifications with four activities in development and maintenance phases, i.e., forward engineering as a development activity, re-engineering, corrective engineering, and management as maintenance activities. They defined some keywords for each classification: keywords were searched and a commit was classified as soon as any keyword had been found in that commit message.

Yamauchi et al. [33] proposed a new method to cluster commits. Their approach detected identifier names related to changes in each commit and classified each commit via the bag-of-words model with the identifier names. Identifiers names for each commit were extracted from syntax differences. Their pilot study confirmed the usefulness of their approach.

3. Methodology

In this section, we introduce in detail the methodology we have followed in our research. The overall procedure consists of 7 steps as shown in Figure 1: the first one comprises all the operations to construct the dataset made up of commit messages from gitHub; the second step, called data preprocessing, includes all the necessary operations to clean the commit messages and remove non-significant terms; the third step comprehends the extraction of features that have been filtered in the fourth step; in the fifth step we have applied some clustering techniques on the features extracted; then, in the sixth step, once obtained the labels we have applied several ML-based classification techniques; and produced our results in the seventh step.

4. Database Construction

The considered software projects are all belonging to the High Energy Physics (HEP) domain and have been mostly developed in the C++ language. We have extracted commit messages from 167 projects belonging to HEP experiment repositories: ALISW [34] contributed with 102 projects, LHCB [35] with 107 projects, CMS-SW [36] provided 41 projects

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

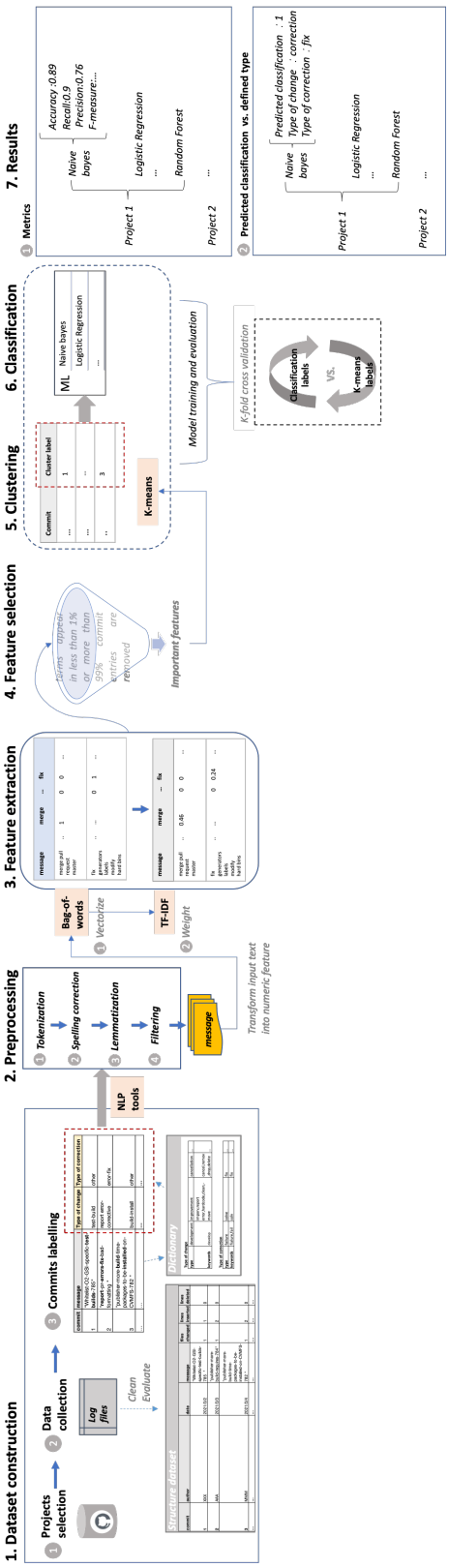

Figure 1. Methodology Overview.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

and, finally ROOT [37] provided 1 project. At the end we have kept 65 software projects (detailed in Table 1) with a number of commit entries higher than 100.

Table 1. Projects in ALISW, LHCb, CMS-SW, ROOT.

Experiment Name		Project Names	
ROOT		root	
LHCb	Condorcet	DevelopKit	analysis-essentials
	bender-tutorials	developkit-lessons	first-analysis-steps
	opendata-project	second-analysis-steps	starterkit-lessons
CMS-SW	cmssw	cms-sw.github.io	cmssw-config
	cms-bot	cms-docker	genproductions
	SCRAM	root	cmssdt-web
	cmssdt-ib	pkgtools	hlt-confdb
	cms-git-tools	jenkins-backup	Stitched
	web-confdb	cmssw-framework	apt-rpm
	apt-rpm	int-build	RecoLuminosity-LumiDB
	DQM-Integration	cmspkg	
ALISW	AliDPG	AliPhysics	AliRoot
	CMake	FairRoot	KFParticle
	RootUnfold	Vc	ali-bot
	alibuild	alidist	arrow
	aurora	cctools	clhep
	cpython	create-pull-request	delphes
	docks	gcc	geant3
	geant3-oldfork	geant4$_v mc$	grpc
	gsl	liblzma	libpng
	libpng-old	mesos-plugin	rapidjson
	release-validation	root	vault-gatekeeper-mesos

The commit message collection was conducted till March 2021 and have been obtained by the *git log* command. Figure 2 shows examples of original unstructured commit information.

Figure 2. Original commit data.

Each unstructured commit has been processed by extracting its meaningful features, such as the name of author, date, the body of the message, the number of changed files, the number of inserted lines and the number of deleted lines. Table 2 shows how the unstructured commit data are turned into structured data for each commit.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

Table 2. Structured dataset example.

Commit ID	Author	Date	Commit Message	N. Files Changed	N. Lines Inserted	N. Lines Deleted
1	XXX	2021/3/2	Whitelist-O2-GSI-specific-test-builds-785	1	1	0
2	AAA	2021/3/3	publisher-more-build-requires-784	1	2	0
3	MMM	2021/3/4	publisher-more-build-time-packages-to-be-installed-on-CVMFS-782	1	2	0

Furthermore, once obtained the structured data, we have gathered some statistics concerning the commit messages, their projects and project groups, such as the total number of days in which the commit messages have been pushed, the numbers of developers involved, the total number of commits, the names and the total number of the modified files, the number of authors per commit, the range of project time period [start date, end date]. Figure 3 shows the number of commits per project: 13 projects have more than 10000 commits, 12 projects have commits between 2000 to 10000, 40 projects have commits between 100 to 2000. The sample size of these projects is adequate for training clustering and classification models.

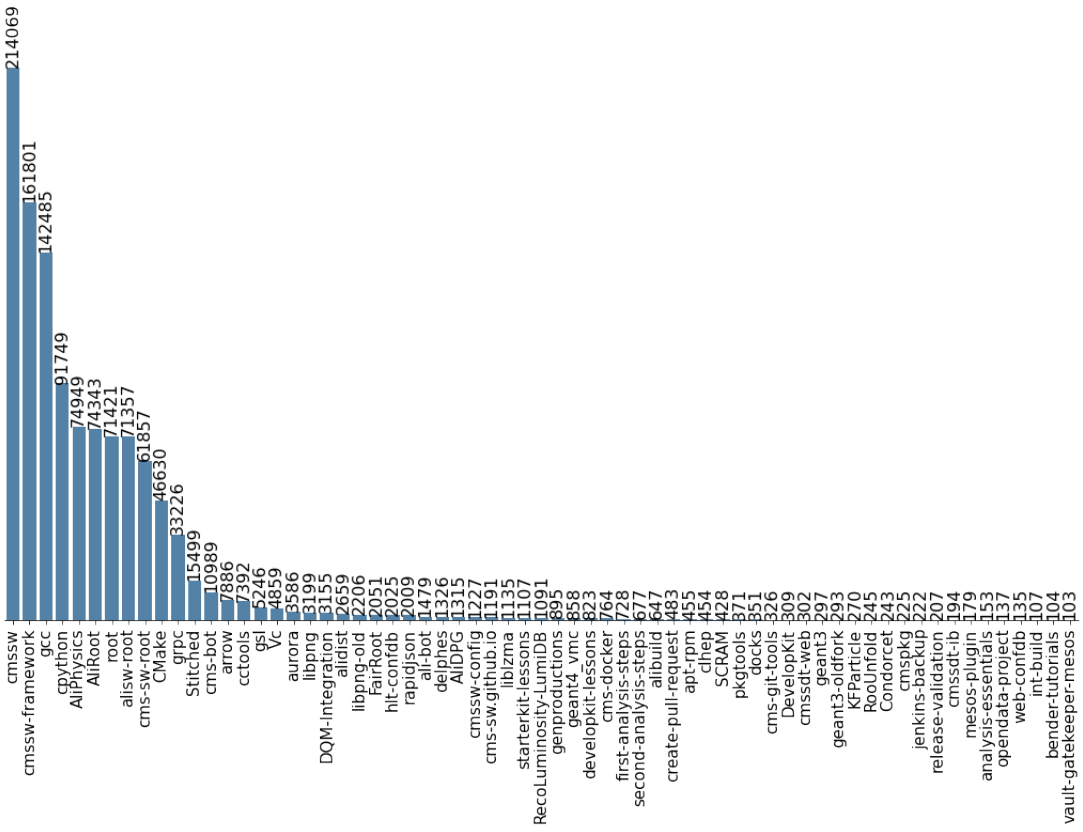

Figure 3. Number of commits per project.

Based on previous studies [38–41] and the knowledge of experts in software engineering, we have identified categories of types of commit. As regards the available categories, we have used a previous taxonomy to which we have added other group types by extracting key terms from the commit messages analyzed: we have especially widen the vocabulary to define better the type of corrective activity. The *type of changes* and *type of corrections* in the commits (see Table 3) have been identified according to keywords in the commit messages. We have first looked for keywords, and, after, if any keyword is found in the commit message, we have labelled a commit accordingly. The *type of changes* is used

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

to keep trace of general code change activities, while the *type of corrections* is mainly related to software problems.

Table 3. Types of changes and corrections. See [39,40] for types in bold.

Category	Values		
Type of changes	Development	Improvement	Performance
	Optimization	Cancellation	Documentation [42]
	Installation	Deployment	Debugging [42]
	Build [42]	Upgrade	Versioning [42]
	Refactoring [42]	Testing [42]	Feature Addition
	Update	Configuration	Dependency
	Indentation [42]	Initialization [42]	Platform [42]
	Corrective [42]	Branch [42]	Legal [42]
	Merge [42]	Revert	New Functionality
Type of corrections	Bug [42]	Issue	Solve
	Patch [39]	Abort [39]	Error [39]
	Bad [39]	Conflict [39]	Problem [39]
	Wrong [39]	Mistake	Avoid [39]
	Warning [39]	Fix	Anomaly
	Failure Exception	Crash	Incident
	Side Effect	Fail	Defect
	Glitch		

Table 4 shows an example of commits with two categories of types of commit. For each commit, the values of *type of change* or *type of correction* can vary according to the keywords of types that are in the correspondent commit message The *type of change* values of the first commit are *test* and *build* (types are connected with the symbol '-'): *test* matches with the *testing* type; *builds* matches with the *build* type. Some commits have *other* values for *type of change* or *type of correction*, when their commit messages do not match any keywords in the dictionary. In this case, the characteristics of commits cannot be recognized through the dictionary.

Table 4. Data with examples of types.

Commit	Commit Message	Type of Change	Type of Correction
1	"Whitelist-O2-GSI-specific-test-builds-785"	test-build	other
2	"report-pr-errors-fix-bad-formatting "	improvement-corrective	error-fix
3	"publisher-more-build-time-packages -to-be-installed-on-CVMFS-782 "	build-install	other

In this research, we have implemented an automatized labelling procedure based on the two variable *type of change* and *type of correction* with the aim of checking the categories obtained as a result of the ML classification techniques and using the resulting dataset for the ground-truth activity[43].

5. Data Preprocessing

This step has involved the use of NLP techniques to clean the commit messages and to identify their key terms. Figure 4 shows in more detail the steps we have followed.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

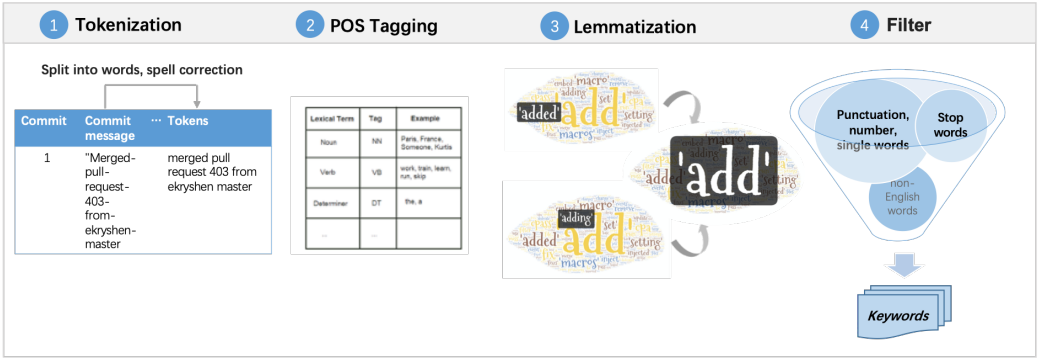

Figure 4. An example of step 2 - Data preprocessing.

Firstly, during the tokenization phase, we have split messages into words correctly spelled; the filtering procedure has included the removal of stop words, punctuation, numbers, non-English words. Furthermore, during the part-of-speech (POS) tagging phase, we have identified nouns, verbs and adjectives to put in input of the lemmatization [21,44] process, which is the process of deriving the lemma, e.g. the base or dictionary form, of a given word due to the existence of different inflected forms. An example of application of the lemmatization process is the one that reduces the two forms "added" and "adding" to the lemma "add" where the inflectional endings, such as "ed" and "ing" have been removed.

This step implied a manual cross check of the key terms found in every commit message to avoid the removal of meaningful terms.

6. Feature Extraction

This step includes the text vectorization of the key terms found in the second step, named data preprocessing. We have used bag of words (BoW) technique that assigns 1 when a certain word is included in a message and 0 otherwise. Afterwards, we have applied the term frequency-inverse document frequency (TF-IDF) technique [41,45] on the BoW terms to transform each text into a numeric feature. TF-IDF is an information retrieval technique that tries to assess how relevant a word (i.e. term) is to a document (commit message) in a collection of documents (log messages): the document concept is represented in our context by a commit message, while the terms "collection of documents" refers to the collection of log messages. Figure 5 shows more in detail how the process of feature extraction works.

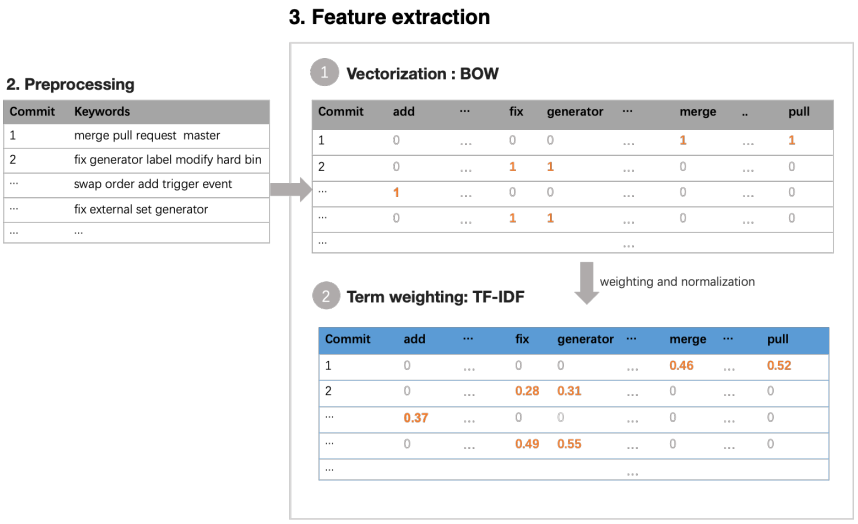

Figure 5. An example of step 3 - Feature extraction.

Equation 1 introduces the value of TF-IDF

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

$$TF - IDF = tf(t,d) \times idf(t) \quad (1)$$

which is the combination of two metrics:

1. tf(t, d) is the term frequency (i.e. the number of times) of the term t in a document d;
2. idf(t) = $log\left(\frac{N}{1+|d \in D : t \in d|}\right)$ 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 $|D|$.

Higher values of TF-IDF entail a higher relevance of a term in a particular document. Figure 6 highlights the TF-IDF outcome of the first commit in Figure 5: the terms merge and pull have assigned low frequency values with respect to the whole collection of log messages.

Commit	add	···	fix	generator	···	merge	···	pull
1	0	...	0	0	...	0.46	...	0.52

Figure 6. TF-IDF outcome.

7. Clustering

In this step we have applied k-means clustering technique. k-means [46–48] is a clustering method that divides n observations into groups so that in each group every observation is closely related. This kind of clustering is the most used because of its less complexity and efficiency. The number of clusters is required in input before execution: to estimate k, the elbow method [49,50] and silhouette coefficient score [51,52] can be used.

7.1. Number of Clusters

Figure 7 shows the number of clusters for all studied projects. They range from 5 to 16: 19 projects have a number of clusters between 5 and 7; 14 projects have 10 to 12 clusters; 11 projects have a number of clusters between 13 to 14; 3 projects have 15 to 16 clusters.

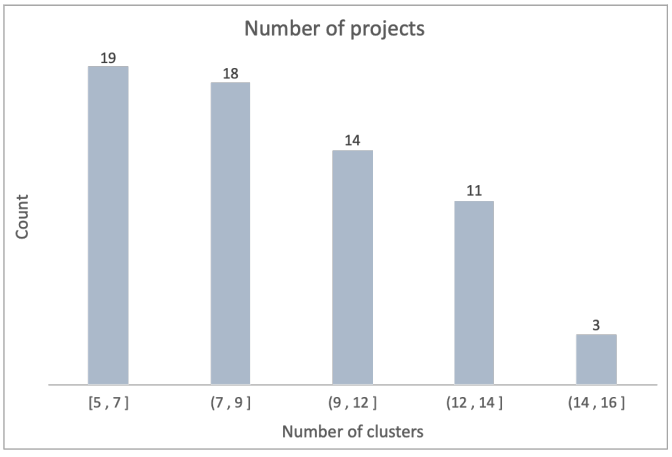

Figure 7. Number of clusters distribution of studied groups of projects (i.e. ALISW, LHCb, CMS-RW, ROOT)

Figure 8 shows the AliDPG project (in ALISW group)'s cluster numbers (i.e. k values) chosen by the elbow method on the left side and the silhouette coefficient score on the right side. Plots report the maximum curvature as the k value: if the line chart resembles an arm, the elbow implies that the underlined model fits best at the point. The k value chosen by the two methods are usually different. In this study, the k value chosen by the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

elbow method has been considered first: if the elbow method does not have the maximum curvature point, the k value recommended by the silhouette coefficient would be used. For the AliDPG project, (see Figure8), the Elbow method suggests k=8 with the sum of squared error within clusters equal to 824.589; the silhouette score also suggests k=8 with an average silhouette coefficient of 0.228. Therefore, k=8 has been set for AliDPG.

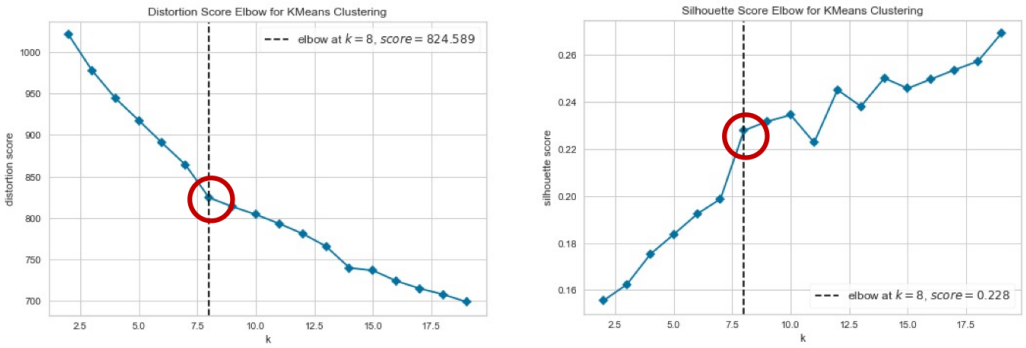

Figure 8. AliDPG - number of clusters. k values with the Elbow method on the left side and the silhouette coefficient score on the right side.

7.2. Clustering evaluation

Only when there are patterns of clusters, the k-means clustering labels are valid and can be used in classification models. Results show that for the majority of projects, patterns appear in k-means clusters and each cluster is mainly dominated by one combination of the *type of change* and *type of correction*. This is a promising result, which means k-means labels can be considered as commits labels and apply to the following classification models.

In the following, considerations are given for the AliDPG project in the ALISW group. The number of commits assigned to each cluster has been evaluated. Table 5 shows the distribution of clusters for the AliDPG project: the cluster 0 has the most commits with 36.05% commits and cluster 7 has the least commits with 3.35% commits. There are 15.29% commits in cluster 1, 6.24% commits in cluster 2, 8.25% commits in cluster 3, 11.10% commits in cluster 4, 7.45% commits in cluster 5, 12.02% commits in cluster 6.

Table 5. AliDPG - distribution of clusters.

cluster label	number of commits	percentage of commits
0	474	36.05%
1	201	15.29%
2	82	6.24%
3	112	8.52%
4	146	11.10%
5	98	7.45%
6	158	12.02%
7	44	3.35%

Before observing the patterns of clusters, the distribution of *Type of change* and *Type of correction* variables of AliDPG have been analyzed. Combing the two variables, Table 6 shows types with the percentage of commits above 1%. According to the identification of types (detailed in section 4), 23.35% of commits have *other* value for *type of change* and *type of correction*, which means their types have not been identified by the defined dictionary. k-means clustering can help assign these undefined commits into different clusters and assign the label for each commit. The second most common *type of change* is *merge* without any correction type, accounting for 14.9% of commits. The third most common *type of*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

change is *feature addition* without any correction type, accounting for 14.1% of commits. The following common *type of change* is *corrective*, which means *type of correction* has been identified (*fix*), having 6.84% of commits. During this process, the idea that *k*-means clustering can help to assign those undefined commits into different clusters and help to explore more comprehensive patterns among commits, has been confirmed.

Table 6. AliDPG - distribution of types.

Type of change	Type of correction	Number of commits	Percentage of commits
other	other	307	23.35%
merge	other	196	14.90%
feature addition	other	186	14.14%
corrective	fix	90	6.84%
configuration	other	63	4.79%
update	other	61	4.64%
branch-merge	other	45	3.42%
cancellation	other	37	2.81%
corrective	bug-fix	24	1.83%
configuration - feature addition	other	21	1.60%
Internal or ThirdParty dependency	other	17	1.29%
corrective	fault	14	1.06%
versioning	other	14	1.06%
...	...	...	...

For the majority of projects, each cluster is usually dominated by one combination of *type of change* and *type of correction*. As shown in Table 7, the majority of undefined commits have been assigned to cluster 0 where 227 commits have *other* type. Cluster 1 is dominated by *merge* without some specific correction, having 186 commits in it. Cluster 2 is dominated by *update* and *update* with *configuration*. Cluster 3 is dominated by the undefined type of change and type of correction but some commits also involve in *feature addition* (with 16 commits) or *fix* (with 9 commits). Cluster 4 is dominated by *fix*, *bug* and *fix*, and this cluster is mainly related to correction. Cluster 5 is dominated by *feature addition* (with 36 commits) and undefined commits (with 35 commits). Cluster 6 is dominated by *feature addition* and some commits are also related to *configuration*, *revert*, *merge*. Cluster 7 only contains commits that have the change combination of *branch* and *merge* without any correction.

Overall, the clustering has been used to identify potential patterns among commit entries and to evaluate how the *type of change* and *type of correction* are distributed in the commit entries. *k*-means clustering labels are used to classify the commit entries in the classification phase.

8. Classification

In this step we have applied a set of ML techniques to classify commit messages. These are: Naive Bayes, Multinomial Naive Bayes, Random Forest, Decision Tree, Bagging, Logistic Regression, AdaBoost. As for the Bagging method, it relies on Decision Tree models as base classifier thus its results can be compared with tree-based methods [53](e.g., Decision Tree and Random Forest).

This study has adopted the *K*-fold cross validation method in the model evaluation phase. Cross validation is used to evaluate the performance of machine learning models [54]. This method ensures that the score of the model is not affected by the way researchers choose the training and test sets. Cross validation split dataset D into K disjoint subsets: $D = D1 \cup D_2 \cup \ ... \cup D_K = \ (i \neq j)$. It makes the data distribution of each subset D_i be the same as possible. It takes $K-1$ subsets as a training dataset and the remaining subset as a test dataset, and repeats the procedure K times, eventually return the mean values of K testing results. A key parameter of the K-fold cross validation is K (i.e., the number of fold of a given dataset). Some choices are $K = 3$, $K = 5$, and $K = 10$, and the most widely used value in model evaluation is 10. $K = 10$ is better for large data size [55] and it provides a

Table 7. Type of commits per cluster.

Cluster Number	Type of Change	Type of Correction	Number of Commits
Cluster 0	**other**	**other**	**227**
	configuration	**other**	**52**
	cancellation	other	31
	feature addition	other	22
	Internal or ThirdParty dependency	other	14
	corrective	fault	13
	corrective	bug-fix	10
	...	...	...
Cluster 1	**merge**	**other**	**186**
	corrective-merge	patch	5
	corrective-merge	fix	3
	...	...	...
Cluster 2	**update**	**other**	**56**
	configuration-update	other	10
	cancellation-corrective-feature addition-update	fault	2
	cancellation-update	other	2
	configuration-corrective-update	fault	2
	merge-update	other	2
	update-versioning	other	2
	...	...	...
Cluster 3	**other**	**other**	**40**
	feature addition	other	16
	corrective	fix	9
	cancellation	other	5
	...	...	...
Cluster 4	**other**	**other**	**40**
	feature addition	other	16
	corrective	fix	9
	cancellation	other	5
	versioning	other	4
	update	other	4
	configuration	other	4
	...	...	...
Cluster 5	**feature addition**	**other**	**36**
	other	**other**	**35**
	configuration	other	7
	corrective	fix	4
	...	...	...
Cluster 6	**feature addition**	**other**	**111**
	configuration-feature addition	other	19
	feature addition-revert	other	5
	feature addition-merge	other	4
	cancellation-feature addition	other	3
	...	...	...
Cluster 7	**branch-merge**	**other**	**44**

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

good trade-off between computational cost and bias. Therefore, 10-fold cross validation has been used in this study.

Our model evaluation is based on different performance metrics, such as accuracy, precision, recall and F1-score. For binary classification problems see Equations 2-5:

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{2}$$

$$precision = \frac{tp}{tp + fp} \tag{3}$$

$$recall = \frac{tp}{tp + fn} \tag{4}$$

$$F1 - score = 2 \times \frac{precisionxrecall}{precision + recall} \tag{5}$$

where tp is the number of true positive, tn is the number of true negative, fn is the number of false negative and fp is the number of false positive.

In this study, we have faced a multi-class classification problem. We have computed the overall performance metrics as the average of the different binary problems that constitute the multi-class problem [53,56]. To achieve this purpose both macro-average and micro-average techniques have been used; the obtained results may differ according to the method involved. Macro-averaged techniques are more suitable to emphasize the ability of a classifier to behave well also on categories with few examples. In addition, macro-averaged methods give to all classes identical weights, whereas in micro-average equal weights correspond to per-document classification decision [56]. In our context, few samples per category is the most common scenario, as a consequence, macro average is more used. The employed performance metrics to assess model are macro-precision (macro.P), macro-recall (macro.R) and macro-F1 (macro.F1) as summarized in Equations 6-8:

$$Precision : macro.P = \frac{1}{n} \sum_{i=1}^{N} P_i \tag{6}$$

$$Recall : macro.R = \frac{1}{n} \sum_{i=1}^{N} R_i \tag{7}$$

$$F1 - score : macro.F1 = 2 \times \frac{macro.P \times macro.R}{macro.P + macro.R} \tag{8}$$

where n is the number of classes; P_i represents the precision of different classes, and R_i is the recall of the different classes.

9. Results

In this work we have addressed the following research questions.

RQ1: What has been the impact of NLP techniques on the data preparation phase? By applying NLP techniques in the data preparation phase (i.e. data preprocessing, feature extraction, feature selection), the size of the feature matrix has been reduced. The average reduction percentage of all used projects is 83%. Figure 9 shows the feature reduction percentage per project:

feature reduction percentage per project = (number of features before NLP - number of features after NLP)/ (number of features before NLP)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

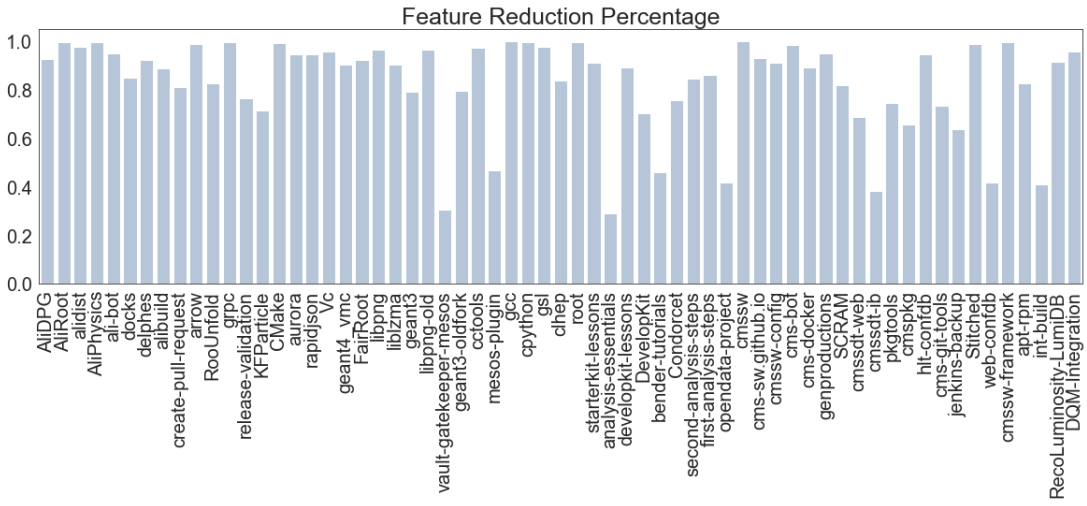

Figure 9. Feature reduction percentage for studied projects (i.e. ALISW, LHCb, CMS-SW, ROOT).

Taken AliDPG as an example, 93% features (all words in original commit messages) have been removed, only 7% features have been used in clustering and classification phases. It helps to remove unimportant and redundant words and only considers important keywords as features. In addition, NLP techniques have an important impact on this study because the F1-scores have improved. The average F1-score of all examined projects has reached 0.9360 and the maximum F1-score has achieved 0.9496. Compared to the previous study [57] that without NLP techniques, in which the maximum F1-score achieved to 0.8210.

RQ2: Which ML classification techniques have the best performance of classifying new commits?

As shown in Table 8, Random Forest, Bagging, Decision Tree have the highest average metric scores (i.e. accuracy, recall, precision, F1-score).

Table 8. The average metric scores of studies projects.

ML Techniques	Accuracy	Recall	Precision	F1-score
AdaBoost (AB)	0.6377	0.4846	0.4359	0.4482
Bagging (BG)	**0.9519**	**0.9280**	**0.9337**	**0.9247**
Decition Tree (DT)	**0.9517**	**0.9295**	**0.9345**	**0.9262**
Logistic Regression (LR)	0.9027	0.8268	0.8810	0.8401
Multinomial Naive Bayes (MNB)	0.8454	0.7487	0.8252	0.7695
Naive Bayes (NB)	0.4979	0.6066	0.5252	0.4964
Random Forest (RF)	**0.9590**	**0.9382**	**0.9448**	**0.9360**

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

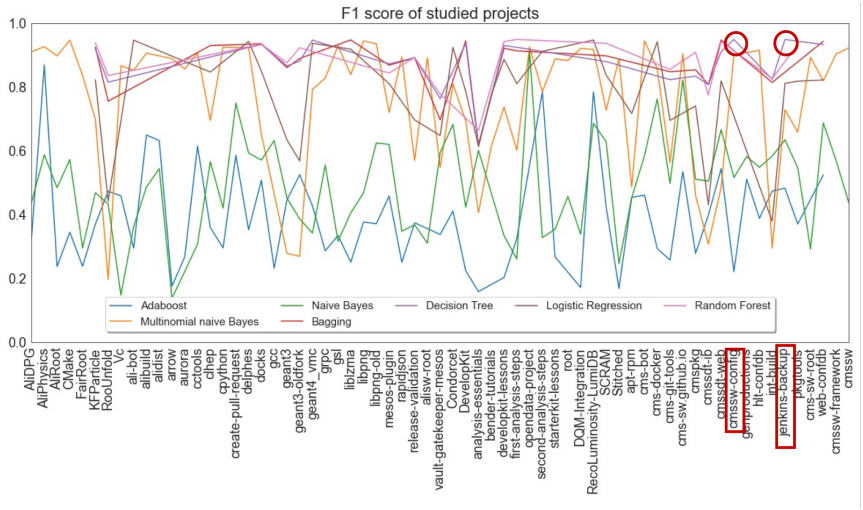

Figure 10. F1-score per project.

In addition, from the perspective of all single projects, cmssw-config and jenkins-backup in CMS-SW project group have reached the highest F1-score (i.e. 0.9496) by Decision Tree algorithm as shown in Figure 10.

RQ3: Do the models behave differently for different projects?

The performance of AdaBoost, Logistic Regression, Multinomial Naive Bayes, and Naive Bayes methods shows huge variations for different projects. The performance of Bagging, Decision Tree, and Random Forest shows stability for different projects.

Table 9. The statistics of metric scores.

Metric score	ML techniques	mean	std	CV
Accuracy	AB	0.6377	0.1712	26.85%
	BG	0.9519	0.0506	5.32%
	DT	0.9517	0.0497	5.22%
	LR	0.9027	0.0959	10.63%
	MNB	0.8454	0.1103	13.05%
	NB	0.4979	0.1827	36.70%
	RF	0.9590	0.0461	4.81%
Recall	AB	0.4846	0.2151	44.40%
	BG	0.9280	0.0760	8.19%
	DT	0.9295	0.0699	7.52%
	LR	0.8268	0.1565	18.92%
	MNB	0.7487	0.1908	25.49%
	NB	0.6066	0.1527	25.18%
	RF	0.9382	0.0691	7.37%
Precision	AB	0.436	0.232	53.27%
	BG	0.934	0.073	7.85%
	DT	0.935	0.068	7.27%
	LR	0.881	0.149	16.86%
	MNB	0.825	0.204	24.74%
	NB	0.525	0.146	27.89%
	RF	0.945	0.063	6.63%
F1 score	AB	0.4482	0.2265	50.54%
	BG	0.9247	0.0792	8.56%
	DT	0.9262	0.0736	7.94%
	LR	0.8401	0.1584	18.85%
	MNB	0.7695	0.2027	26.34%
	NB	0.4964	0.1687	33.97%
	RF	0.9360	0.0706	7.54%

 doi:10.20944/preprints202209.0309.v1

In this study, the Coefficient of Variation (CV) of metric score per ML technique has been considered to measure the stability and variation of model performance. CV is the ratio of the standard deviation (std) to the mean of a random variable and it is independent of the unit of measurement used for the variable. As shown in Table 9, CV values of Bagging, Decision Tree, and Random Forests for different metric scores are smaller than 10%, which implies model stability for different projects. CV values of AdaBoost, Logistic Regression, Multinomial Naive Bayes, and Naive Bayes are higher than 15%, which implies huge variations of the metric scores for different projects.

In addition to the CV values, the distribution of metric scores has been examined. Figures 11, 12, 13, and 14 show the boxplots of metric scores for different ML techniques. For the studied projects, AdaBoost, Logistic Regression, Multinomial Naive Bayes, and Naive Bayes show great variations for accuracy, recall, precision, and F1-score. In their boxplots, the differences between the first quartile and the third quartile (i.e IQR) are large.

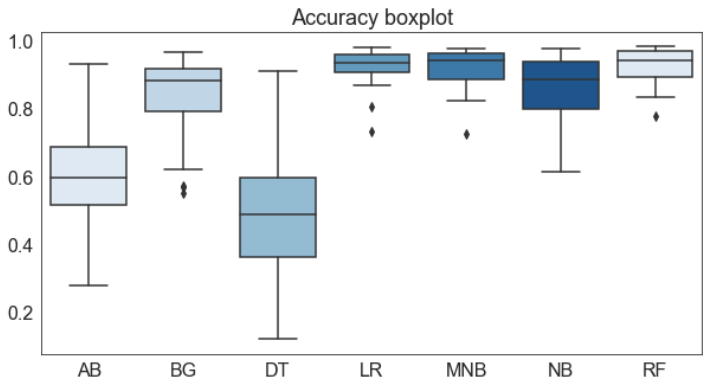

Figure 11. Accuracy Boxplot

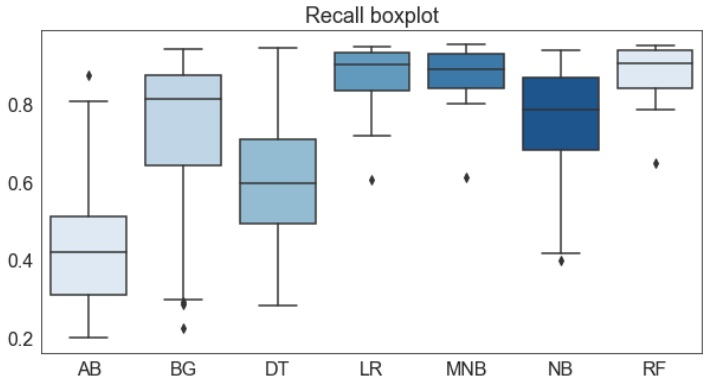

Figure 12. Recall Boxplot

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0309.v1

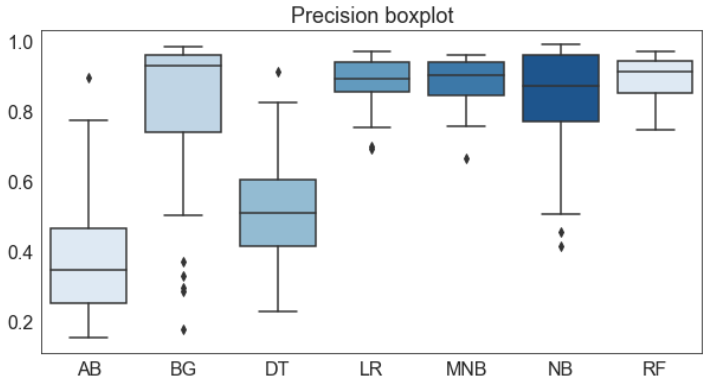

Figure 13. Precision Boxplot

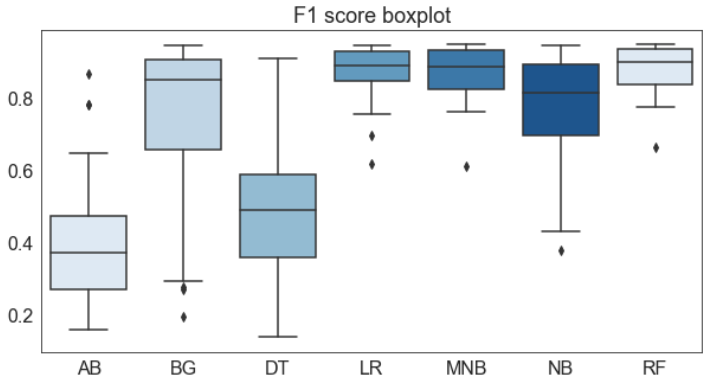

Figure 14. F1 score Boxplot

10. Validity

There are some potential threats to validity. The first one is the diversity of language habits may lead to different data distribution. Authors of commits may have different language habits, so the style of commit messages may be different. They may use different languages to express the same meaning. On average, there were 68 authors per project. So there may be some uncertainty in the original data.

The second one is that only HEP projects stored on GitHub have been studied, and this may have led to a project bias. To overcome this problem, it could be useful to apply the model to software projects in other fields.

The last threat is due to the different sample sizes per project. This may result in projects with huge sample sizes having some advantages to reach better results. To overcome this limitation, the sample size has been taken into account during the result analysis phase.

11. Conclusions

The study is motivated by the demand for analyzing commit messages of open source software on source code management system to monitor the evolution of software and identify code change activities related to software problems.

This study has achieved a promising result. Firstly, the unstructured data has been well manipulated by NLP prepossessing tools and TF-IDF method. Key information in commit messages has been extracted and contributed to clustering and classifying commits. The average feature reduction percentage is 0.83%, and the maximum value of F1-score compared to the previous research [57] has improved. Secondly, code change patterns have been identified by *k*-means clustering, which allows a better understanding of software evolution activities. Finally, ML Classification models used in this study provide a useful

tool to classify new commits into different categories which connect to different types of code change and correction activities.

Concerning the performance of different classification models, this study has validated the robustness of tree-based models on text classification analysis. Metric scores (i.e. accuracy, recall, precision, F1-score) show that models based on trees (Decision Tree, Random Forest, Bagging) have outperformed other models: Random Forest has the highest average score of accuracy (0.9590), recall (0.9382), precision (0.9448) and F1-score (0.9360); Decision Tree has reached the maximum F1-score of 0.9497.

For future work, more clustering methods will be utilized to explore patterns of code changes. In addition, more information of commit activities (e.g., changed files, lines of changed code, inserted code, deleted code) will be considered. This may help to obtain more satisfying results of clustering and classification for code change activities.

Author Contributions: For Conceptualization, E.R. and Y.Y.; methodology, E.R. and Y.Y.; software, Y.Y.; validation, Y.Y., M.C. and E.R.; formal analysis, E.R. and Y.Y.; investigation, Y.Y.; data curation, Y.Y.; writing—original draft preparation, E. R. and M.C.; writing—review and editing, E.E. and M.C.; visualization, Y.Y.; supervision, E.R.; project administration, Y.Y.. All authors have read and agreed to the published version of the manuscript.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: Not applicable

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Constantinou, E.; Kapitsaki, G.M. Identifying Developers' Expertise in Social Coding Platforms. In Proceedings of the 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2016, pp. 63–67. https://doi.org/10.1109/SEAA.2016.18.
2. Thung, F.; Bissyandé, T.F.; Lo, D.; Jiang, L. Network Structure of Social Coding in GitHub. In Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering, 2013, pp. 323–326. https://doi.org/10.1109/CSMR.2013.41.
3. Sarwar, M.U.; Zafar, S.; Mkaouer, M.W.; Walia, G.S.; Malik, M.Z. Multi-label Classification of Commit Messages using Transfer Learning. In Proceedings of the 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2020, pp. 37–42. https://doi.org/10.1109/ISSREW51248.2020.00034.
4. Bavota, G. Mining unstructured data in software repositories: Current & future trends. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016* **2016**, *2016-Janua*, 1–12. https://doi.org/10.1109/SANER.2016.47.
5. Jiang, S.; Armaly, A.; McMillan, C. Automatically generating commit messages from diffs using neural machine translation. In Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2017, pp. 135–146. https://doi.org/10.1109/ASE.2017.8115626.
6. Jalote, P. *An Integrated Approach to Software Engineering*; Texts in Computer Science, Springer, Boston, MA, 2005. https://doi.org/10.1007/0-387-28132-0.
7. Yalla, P.; Sharma, N. Integrating Natural Language Processing and Software Engineering. *International Journal of Software Engineering and Its Applications* **2015**, *9*, 127–136. https://doi.org/10.14257/ijseia.2015.9.11.12.
8. Venigalla, A.S.M.; Chimalakonda, S. Understanding Emotions of Developer Community Towards Software Documentation, 2021.
9. Garousi, V.; Bauer, S.; Felderer, M. NLP-assisted software testing: A systematic mapping of the literature. *Information and Software Technology* **2020**, *126*, 106321. https://doi.org/https://doi.org/10.1016/j.infsof.2020.106321.
10. Siow, J.; Gao, C.; Fan, L.; Chen, S.; Liu, Y. CORE: Automating Review Recommendation for Code Changes. In Proceedings of the 27th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019.
11. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J.; Ajagbe, M.A.; Chioasca, E.V.; Batista-Navarro, R.T. Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study, 2020, [arXiv:cs.SE/2004.01099].
12. Ye, X.; Bunescu, R.; Liu, C. Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation. *IEEE Transactions on Software Engineering* **2016**, *42*, 379–402. https://doi.org/10.1109/TSE.2015.2479232.
13. Gilson, F.; Weyns, D. When Natural Language Processing Jumps into Collaborative Software Engineering. In Proceedings of the 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), 2019, pp. 238–241. https://doi.org/10.1109/ICSA-C.2019.00049.

14. Catolino, G.; Ferrucci, F. Ensemble techniques for software change prediction: A preliminary investigation. In Proceedings of the 2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), 2018, pp. 25–30. https://doi.org/10.1109/MALTESQUE.2018.8368455.
15. Catolino, G.; Palomba, F.; De Lucia, A.; Ferrucci, F.; Zaidman, A. Enhancing change prediction models using developer-related factors. *Journal of Systems and Software* **2018**, *143*, 14–28. https://doi.org/https://doi.org/10.1016/j.jss.2018.05.003.
16. Zhou, Y.; Leung, H.; Xu, B. Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness. *IEEE Transactions on Software Engineering* **2009**, *35*, 607–623. https://doi.org/10.1109/TSE.2009.32.
17. Pritam, N.; Khari, M.; Hoang Son, L.; Kumar, R.; Jha, S.; Priyadarshini, I.; Abdel-Basset, M.; Viet Long, H. Assessment of Code Smell for Predicting Class Change Proneness Using Machine Learning. *IEEE Access* **2019**, *7*, 37414–37425. https://doi.org/10.1109/ACCESS.2019.2905133.
18. Piris, Y.; Gay, A.C. Customer satisfaction and natural language processing. *Journal of Business Research* **2021**, *124*, 264–271. https://doi.org/10.1016/j.jbusres.2020.11.065.
19. Garousi, V.; Bauer, S.; Felderer, M. NLP-assisted software testing: A systematic mapping of the literature. *Information and Software Technology* **2020**, *126*, [1806.00696]. https://doi.org/10.1016/j.infsof.2020.106321.
20. Patil, L.H.; Atique, M. A Novel Approach for Feature Selection Method TF- IDF in Document Clustering. *IEEE International Advance Computing Conference (IACC)* **2012**, pp. 858–862.
21. Ozturkmenoglu, O.; Alpkocak, A. Comparison of different lemmatization approaches for information retrieval on Turkish text collection. In Proceedings of the 2012 International Symposium on Innovations in Intelligent Systems and Applications, 2012, pp. 1–5. https://doi.org/10.1109/INISTA.2012.6246934.
22. Dönmez, P. Introduction to Machine Learning. *Natural Language Engineering* **2013**, *19*, 285–288. https://doi.org/10.1017/s1351324912000290.
23. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. AAAI Press, 1996, pp. 226–231. https://doi.org/10.5555/3001460.3001507.
24. Liang, X.; Hong, T.; Qiping, S.G. Occupancy data analytics and prediction : a case study. *Berkeley Lab* **2016**.
25. Daniel Jurafsky, J.H.M. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*; PEARSON, 2008; pp. 1–306. https://doi.org/10.4324/9781315845593.
26. Yan, M.; Zhang, X.; Liu, C.; Xu, L.; Yang, M.; Yang, D. Automated change-prone class prediction on unlabeled dataset using unsupervised method. *Information and Software Technology* **2017**, *92*, 1–16. https://doi.org/10.1016/j.infsof.2017.07.003.
27. Levin, S.; Aviv, T.; Aviv, T. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. *arXiv* **2017**, [arXiv:1711.05340v1].
28. Messaoud, M.B. On the Classification of Software Change Messages using Multi-label Active Learning **2019**. https://doi.org/10.1145/3297280.3297452.
29. Barnett, J.G.; Gathuru, C.K.; Soldano, L.S.; McIntosh, S. The relationship between commit message detail and defect proneness in Java projects on GitHub. *Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016* **2016**, pp. 496–499. https://doi.org/10.1145/2901739.2903496.
30. Levin, S.; Yehudai, A. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. In Proceedings of the Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering; Association for Computing Machinery: New York, NY, USA, 2017; PROMISE, pp. 97–106. https://doi.org/10.1145/3127005.3127016.
31. Zhong, S.; Khoshgoftaar, T.M.; Seliya, N. Unsupervised learning for expert-based software quality estimation. In Proceedings of the Eighth IEEE Internation Symposium on High Assurance Systems Engineering. IEEE, (2004). https://doi.org/10.1109/HASE.2004.1281739.
32. Lile P. Hattori and Michele Lanza. On the nature of the nature of law. *Archiv fur Rechts- und Sozialphilosophie* **2012**, *98*, 457–467. https://doi.org/10.2139/ssrn.1836494.
33. Yamauchi, K.; Yang, J.; Hotta, K.; Higo, Y.; Kusumoto, S. Clustering commits for understanding the intents of implementation. *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014* **2014**, pp. 406–410. https://doi.org/10.1109/ICSME.2014.63.
34. ALISW. https://github.com/alisw, 2022.
35. LHCB. https://github.com/lhcb, 2022.
36. CMS-SW. https://github.com/cms-sw, 2022.
37. ROOT. https://github.com/root-project/root, 2022.
38. Swanson, E.B. The dimensions of maintenance. *Proceedings - International Conference on Software Engineering* **1976**, pp. 492–497.
39. Hindle, A.; German, D.M.; Holt, R. What Do Large Commits Tell Us? A Taxonomical Study of Large Commits. In Proceedings of the Proceedings of the 2008 International Working Conference on Mining Software Repositories; Association for Computing Machinery: New York, NY, USA, 2008; MSR '08, p. 99–108. https://doi.org/10.1145/1370750.1370773.
40. Khairul Islam, M.; Ahmed, T.; Ahmed, F.; Iqbal, A. Accepted or abandoned? Predicting the fate of code changes. *arXiv* **2019**, [1912.03437].

41. Jiang, Z.; Gao, B.; He, Y.; Han, Y.; Doyle, P.; Zhu, Q. Text Classification Using Novel Term Weighting Scheme-Based Improved TF-IDF for Internet Media Reports. *Mathematical Problems in Engineering* **2021**, *2021*. https://doi.org/https://doi.org/10.1155/2021/6619088.
42. Gharbi, S.; Mkaouer, M.W.; Jenhani, I.; Messaoud, M.B. On the Classification of Software Change Messages Using Multi-Label Active Learning. In Proceedings of the Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing; Association for Computing Machinery: New York, NY, USA, 2019; SAC '19, pp. 1760–1767. https://doi.org/10.1145/3297280.3297452.
43. Golzadeh, M.; Decan, A.; Legay, D.; Mens, T. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software* **2021**, *175*, 110911. https://doi.org/https://doi.org/10.1016/j.jss.2021.110911.
44. Khatiwada, S.; Kelly, M.; Mahmoud, A. STAC: A tool for Static Textual Analysis of Code. In Proceedings of the 2016 IEEE 24th International Conference on Program Comprehension (ICPC), 2016, pp. 1–3. https://doi.org/10.1109/ICPC.2016.7503746.
45. Patil, L.H.; Atique, M. A novel approach for feature selection method TF-IDF in document clustering. In Proceedings of the 2013 3rd IEEE International Advance Computing Conference (IACC), 2013, pp. 858–862. https://doi.org/10.1109/IAdCC.2013.6514339.
46. Ali, A.; Bin Faheem, Z.; Waseem, M.; Draz, U.; Safdar, Z.; Hussain, S.; Yaseen, S. Systematic Review: A State of Art ML Based Clustering Algorithms for Data Mining. In Proceedings of the 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020, pp. 1–6. https://doi.org/10.1109/INMIC50486.2020.9318060.
47. Kapil, S.; Chawla, M.; Ansari, M.D. On K-means data clustering algorithm with genetic algorithm. In Proceedings of the 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), 2016, pp. 202–206. https://doi.org/10.1109/PDGC.2016.7913145.
48. Alsarhan, Q.; Ahmed, B.S.; Bures, M.; Zamli, K.Z. Software Module Clustering: An In-Depth Literature Analysis. *IEEE Transactions on Software Engineering* **2020**. https://doi.org/10.1109/TSE.2020.3042553.
49. Nainggolan, R.; Perangin-Angin, R.; Simarmata, E.; Tarigan, A.F. Improved the Performance of the K-Means Cluster Using the Sum of Squared Error (SSE) optimized by using the Elbow Method. *Journal of Physics: Conference Series* **2019**, *1361*. https://doi.org/10.1088/1742-6596/1361/1/012015.
50. Yuan, C.; Yang, H. Research on K-Value Selection Method of K-Means Clustering Algorithm. *J* **2019**, *2*, 226–235. https://doi.org/10.3390/j2020016.
51. Kaoungku, N.; Suksut, K.; Chanklan, R.; Kerdprasop, K.; Kerdprasop, N. The silhouette width criterion for clustering and association mining to select image features. *International Journal of Machine Learning and Computing* **2018**, *8*, 69–73. https://doi.org/10.18178/ijmlc.2018.8.1.665.
52. Berkhin, P. Clustering survey Bherkin. *Technical Report, Accrue Software* **2002**, pp. 1–56.
53. Sutton, C.D. Classification and Regression Trees, Bagging, and Boosting. In *Data Mining and Data Visualization*; Rao, C.; Wegman, E.; Solka, J., Eds.; Elsevier, 2005; Vol. 24, *Handbook of Statistics*, pp. 303–329. https://doi.org/https://doi.org/10.1016/S0169-7161(04)24011-1.
54. Browne, M.W. Cross-Validation Methods. *Journal of Mathematical Psychology* **2000**, *44*, 108–132. https://doi.org/https://doi.org/10.1006/jmps.1999.1279.
55. Oh, S.L.; Jahmunah, V.; Ooi, C.P.; Tan, R.S.; Ciaccio, E.J.; Yamakawa, T.; Tanabe, M.; Kobayashi, M.; Rajendra Acharya, U. Classification of heart sound signals using a novel deep WaveNet model. *Computer Methods and Programs in Biomedicine* **2020**, *196*, 105604. https://doi.org/10.1016/j.cmpb.2020.105604.
56. Lan, M.; Tan, C.L.; Su, J.; Lu, Y. Supervised and Traditional Term Weighting Methods for Automatic Text Categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2009**, *31*, 721–735. https://doi.org/10.1109/TPAMI.2008.110.
57. Ronchieri, E.; Yang, Y.; Canaparo, M.; Costantini, A.; Duma, D.C.; Salomoni, D. A new code change prediction dataset: a case study based on HEP software. In Proceedings of the Under Publication at IEEE NSS MIC 2020, 2020.