applied sciences

IMPACT FACTOR 2.838

CITESCORE 3.7

Article

Natural Language Processing Application on Commit Messages: A Case Study on HEP Software

Yue Yang, Elisabetta Ronchieri and Marco Canaparo

Special Issue
Data Mining and Machine Learning in Multimedia Databases

Edited by
Prof. Dr. Ilaria Bartolini

MDPI

https://doi.org/10.3390/app122110773

applied sciences

Article

Natural Language Processing Application on Commit Messages: A Case Study on HEP Software

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

[1] Department of Statistical Sciences, University of Bologna, 40126 Bologna, Italy
[2] INFN CNAF, 40126 Bologna, Italy
* Correspondence: elisabetta.ronchieri@cnaf.infn.it; Tel.: +39-051-209-5072

Abstract: Version Control and Source Code Management Systems, such as GitHub, contain a large amount of unstructured historical information of software projects. Recent studies have introduced Natural Language Processing (NLP) to help software engineers retrieve information from a very large collection of unstructured data. In this study, we have extended our previous study by increasing our datasets and machine learning and clustering techniques. 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 them as input of different clustering algorithms. Once each entry was labelled, we applied supervised machine learning techniques to build a prediction and classification model. We have developed a machine learning-based model to automatically classify commit messages of a software project. Our model exploits a ground-truth dataset that includes commit messages obtained from various GitHub projects belonging to the High Energy Physics context. The contribution of this paper is two-fold: it proposes a ground-truth database and it provides a machine learning prediction model that automatically identifies the more change-prone areas of code. Our model has obtained a very high average accuracy (0.9590), precision (0.9448), recall (0.9382), and F1-score (0.9360).

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

check for updates

Citation: Yang, Y.; Ronchieri, E.; Canaparo, M. Natural Language Processing Application on Commit Messages: A Case Study on HEP Software. *Appl. Sci.* **2022**, *12*, 10773. https://doi.org/10.3390/app122110773

Academic Editor: Francisco García-Sánchez

Received: 15 September 2022
Accepted: 20 October 2022
Published: 24 October 2022

1. Introduction

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

During software development, all the generated data are stored in Version Control and Source Management System (SCMS) repositories. GitHub is the largest SCMS in the word [3] and contains a wealth of data for each open source project. SCMS 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 project on SCMS, 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].

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 SCMS 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. To simplify this activity, Natural Language Processing (NLP) [6,7] can be considered a possible solution to categorize code changes.

Since all the artifacts produced are plain text, NLP has been exploited 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; they may occur both during the development and maintenance of a software system; they may 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 this study, we have extended the building of the historical code change dataset [18,19], composed of commit messages and classified according to their types of changes and corrections. This dataset can represent a ground-truth dataset for further studies in the context of code change proneness prediction. We have executed systematic queries to monitor the evolution of a software project and designed a multi-label classification model to associate a commit message with more than one class label. With respect to our previous studies [18,19], 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 10 we draw our conclusions.

2. Related Work

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

2.1. NLP Techniques on Commit Messages

NLP is composed of several steps, such as tokenization [20], part-of-speech (POS) tagging [9], lemmatization [21], stemming [22], and stop-words removal [20,22]. They have been applied on commit messages to simplify the text parsing and obtain information on software development process. In the following, we summarize the exploitation of NLP in some studies.

Dos Santos et al. [23] built a labelled dataset composed of commit entries found in previous works. They demonstrate the usefulness of NLP techniques by comparing performance metrics of only ML-based models to those of ML and NLP-based models. Their findings show that an NLP-based model outperformed the first model, achieving 91% of F-measure for the commit classification task.

Nyamawe [24] applies NLP to commit messages with the aim of building a ML-based refactoring recommender, afterwards this new methodology is compared with traditional refactoring detectors over 65 open source projects. The results are encouraging since the commit message-based method shows better performance then others.

Sagar et al. [25] focus on detecting the type of code refactoring performed. The proposed model takes in input both structural information of the code, such as coupling and complexity, and commit messages. After testing 800 Java projects and dataset of 5004 commits, their best findings show that the random forest-based model achieves an average accuracy of 75%.

Rebai et al. [26] propose a framework that performs an NLP-based analysis of commit messages and checks the software quality in the context of software refactorings. The outcome is composed of context-driven refactoring recommendations. The evaluation shows promising results and is carried out on six open-source projects and compares to the state-of-art approaches based on static and dynamic analysis.

Rantala et al. [27] present a study in the context of technical debt. They work on detecting commits that introduce self-admitted technical debt. Their approach exploits NLP methods and is applied to five open-source projects. The authors compare their research with a manually labelled dataset from prior work and show that their approach achieved a +0.15 better area under the ROC curve.

Jiang et al. [5] train a neural machine translation algorithm using pairs of diffs and commit messages from 1k popular projects on GitHub with the aim of creating a framework that builds commit messages starting from diffs. NLP is used to find similar patterns in the commit messages of their datasets.

Jung [28] presents a NLP-based framework that summarizes code modifications to solve the difficulty of humans manually writing commit messages. The author shows that the framework achieves good results in generating commit-messages.

The above-mentioned works employ and describe the NLP techniques we have applied in the second step of our methodology, called data preprocessing. Our purpose is to clean commit messages texts and to remove non-significant terms, to reach our achievements we apply tokenization, POS tagging, lemmatization, and stop-words removal.

2.2. Classification of Commits

Commit messages help to keep track of the changes of software on SCMS (e.g., 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 [3,29–31]. Research on the classification of commit messages is still ongoing.

Barnett et al. [32] consider 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 uses Naive Bayes classifiers to predict the probability of becoming a defective commit. The explanatory power of the JIT models demonstrates a significant improvement for 43–80% of the studied project, and it reaches 72%.

Levin et al. [33] introduce developers' information and add novel metrics that capture temporal and semantic developer-level information. They use a generalized regression modeling (GLM) in the R statistical environment to explore their dataset and build predictive models. They define 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. Their study reaches a promising predictive power with R^2 values of 0.75, 0.83, and 0.64.

Levin et al. [30] extend their research, and utilize 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 reaches a promising accuracy of 76% and Cohen's kappa of 63%.

Gharbi et al. [31] use the term frequency-inverse document frequency (TF-IDF) method to extract useful features (i.e., keywords) from commit message text, and apply active learning to reduce the workload of labelling commit messages. They use multi-label active learning with an auxiliary learner strategy and logistic regression model classifier to set up an iteration mechanism, and the classifier model is trained on the updated training set and tested on a separate testing set. They achieve 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] propose an off-the-shelf neural network, called DistilBERT, and fine-tune it for the commit message classification problem. They consider code-change classifications, e.g., bug fix, feature addition, and performance improvement. These modifications contribute to increasing the complexity of a software system, causing the risk of new defects. They utilize an NLTK package to preprocessing data and mutually labelled commit messages with multi-label. Their model reaches an F1-score of 87%.

Leveraging previous works on commit classification we employ the same techniques in the step of our methodology called extraction of features. In more detail, we use bag of words and TF-IDF techniques with the aim of transforming each commit text into a numeric feature.

2.3. Clustering Analysis of Commits

The above studies consider labelled commits for classification. In reality, labelling commits have many problems: commit submissions are usually huge, so manual labelling involves a lot of work; manual labelling can easily produce bias (e.g., different annotators may assign different labels for the same commit message, influenced by their subjective judgment or individuals' experience); 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. [34] use unsupervised learning for expert-based software quality estimation. They firstly cluster a great number of software modules into some groups and invite qualified experts to label each cluster as fault-prone or not fault-prone. Clustering techniques performed in this study are k-means and neutral gas, and results demonstrate that the neutral-gas algorithm outperforms k-means.

P. Hattori et al. [35] classify commits into four categories. Unlike categories proposed by Swanson's classification of maintenance activities, they propose 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 define some keywords for each classification: keywords are searched and a commit is classified as soon as any keyword has been found in that commit message.

Yamauchi et al. [36] propose a new method to cluster commits. Their approach detects identifier names related to changes in each commit and classifies each commit via the bag-of-words model with the identifier names. Identifiers names for each commit are extracted from syntax differences. Their pilot study confirms the usefulness of their approach.

In our work, after analysing the methods used in previous literature, we selected the k-means clustering technique thanks to its simplicity and efficiency.

2.4. Sentiment Analysis and Clustering Models

Zhang et al. [37] introduce a product ranking model based on sentiment analysis combined with an intuitionistic fuzzy TODIM method. By leveraging a case study on a mobile phone selection, this work illustrates all the steps followed for the product selection. The first step regards product feature extraction by the Apriori algorithm based on online reviews. The second step relies on sentiment analysis to detect the sentiment orientation and intensity. The third step represents the sentiment orientation of the product features by an intuitionistic fuzzy value. Finally, the intuitionistic fuzzy TODIM method is used to determine the ranking results of the alternative products.

Zhou et al. [38] propose a novel approach in the context of online reviews. In their study they describe a feature ranking algorithm that is capable of handling feature extraction and rank problem with diverse impact factors and high uncertainty.

Zhou et al. [39] focus on the relationship between emoji and personality recognition. The authors propose two novel attention-based Bi-LSTM architectures to incorporate emoji and textual information at different semantic levels. Their findings show that the proposed methods are comparable to the state-of-the-art methods in performance over the baseline models on the real dataset, demonstrating the usefulness and contribution of emoji information in personality recognition tasks.

Zhang et al. [40] develop a new clustering method to cluster judgment debtors and analyse their main characteristics. They employed the hesitant fuzzy linguistic term sets (HFLTSs) representing the judgment debtors' attributes. Furthermore, they propose new distance measures of HFLTSs to effectively determine the judgment debtors and discuss their properties. Finally, a new HFL-AHC method for clustering judgment debtors is developed based on new distance measures and compared to previous methods.

The aforementioned works give an insight of some novel approaches of NLP, clustering and feature extraction techniques employed in the context of sentiment analysis.

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 considered software projects are all belonging to the High Energy Physics (HEP) domain and have been mostly developed in the C++ language.

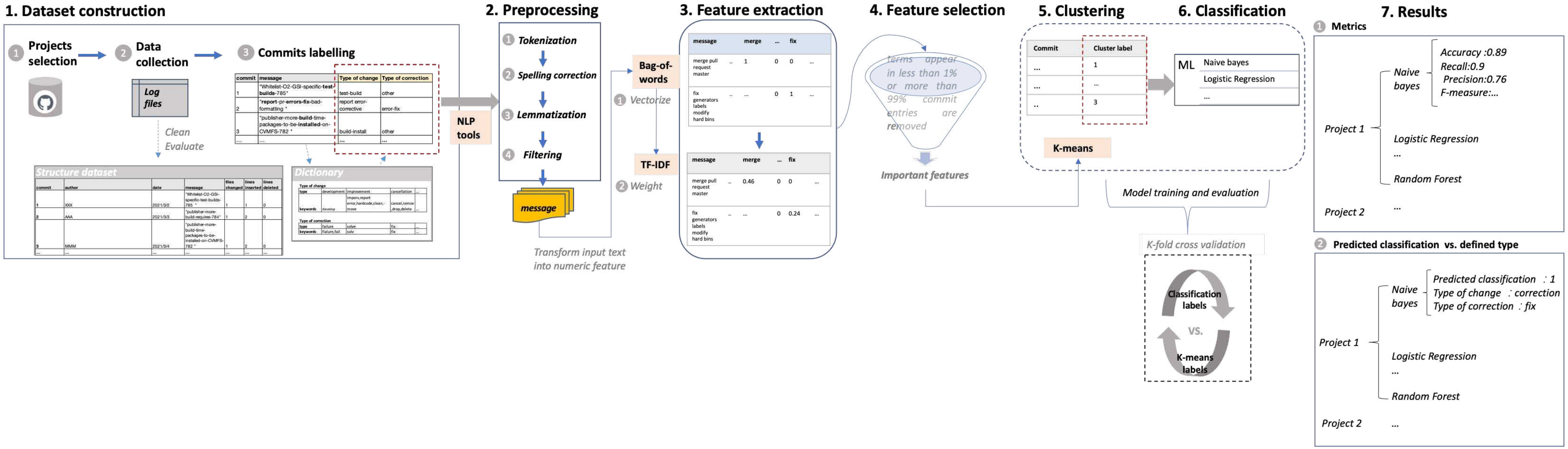


Figure 1. Methodology overview.

The second step, called data preprocessing, includes all the necessary operations to clean the commit messages and remove non-significant terms. This step has involved the use of NLP techniques to clean the commit messages and to identify their key terms:

- Tokenization 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.
- 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. POS tagging helps to identify the syntactic relationships between words to understand the meaning of sentences.
- Lemmatization takes into consideration the morphological analysis of the words, converting the given word to its base form in a dictionary. This approach recognizes the inflectional form of the word and returns its basic form (for example, "are" is reduced to "be").
- Stop-words removal exploits words, such as articles, pronouns, and prepositions, that bring little contextual information, but their occurrence frequency is high. This activity reduces the number of features and the noise in text, and helps to obtain key information.

The third step comprehends the extraction of features that have been filtered in the fourth step. 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 [22,41] 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.

In the fifth step we have applied k-means clustering technique on the features extracted. k-means [42–44] 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 [45,46] and silhouette coefficient score [47,48] can be used.

Then, in the sixth step, once obtained the labels through clustering, we have applied several ML-based classification techniques; and produced our results in the seventh step. In this step, we have applied a set of ML techniques to classify commit messages, as detailed in the following:

- Naive Bayes and Multinomial Naive Bayes techniques that belong to the same model family. The various Naive Bayes classifiers have a different assumption about the likelihood of the features [49]. For example, 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 that is a special generalized linear model that 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 [50].
- Decision Tree that 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 [50].
- Random Forest that 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.

- Bagging, also called bootstrap aggregating, that 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 neighbour, and logistic regression).
- 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. As for the Bagging method, it relies on Decision Tree models as base classifier thus its results can be compared with tree-based methods [51] (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 [52]. 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 [53] and it provides a 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):

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{1}$$

$$precision = \frac{tp}{tp + fp} \tag{2}$$

$$recall = \frac{tp}{tp + fn} \tag{3}$$

$$F1\text{-}score = 2 \times \frac{precision \times recall}{precision + recall} \tag{4}$$

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.

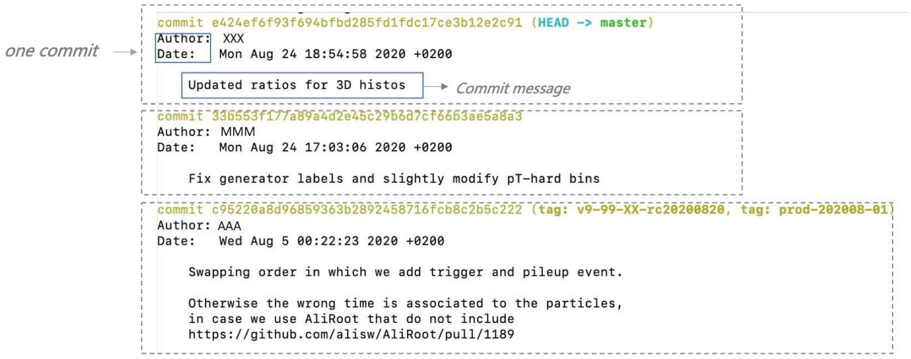
4. Database Construction

We have extracted commit messages from 167 projects belonging to HEP experiment repositories: ALISW [54] contributed with 102 projects, LHCB [55] with 107 projects, CMS-SW [56] provided 41 projects, and, finally, ROOT [57] 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 HEP experiments, i.e., ALISW, LHCb, CMS-SW, and 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_vmc	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.



```
commit e424ef6f93f694bfbd285fd1fdc17ce3b12e2c91 (HEAD -> master)
Author:  XXX
Date:   Mon Aug 24 18:54:58 2020 +0200

    Updated ratios for 3D histos

commit 33b553f177a89a4d2e45c29b6d7cf66b3ae5a8a3
Author: MMM
Date:   Mon Aug 24 17:03:06 2020 +0200

    Fix generator labels and slightly modify pT-hard bins

commit c95220a8d96859363b2892458716fcb8c2b5c222 (tag: v9-99-XX-rc20200820, tag: prod-202008-01)
Author: AAA
Date:   Wed Aug 5 00:22:23 2020 +0200

    Swapping order in which we add trigger and pileup event.

    Otherwise the wrong time is associated to the particles,
    in case we use AliRoot that do not include
    https://github.com/alisw/AliRoot/pull/1189
```

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.

Table 2. Structured dataset example.

Commit ID	Author	Date	Commit Message	N. Files Changed	N. Lines Inserted	N. Lines Deleted
1	XXX	2 March 2021	Whitelist-O2-GSI-specific-test-builds-785	1	1	0
2	AAA	3 March 2021	publisher-more-build-requires-784	1	2	0
3	MMM	4 March 2021	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, and the range of project time period [start date, end date]. Figure 3 shows the number of commits per project: 13 projects have more than 10,000 commits, 12 projects have commits between 2000 to 10,000, 40 projects have commits between 100 to 2000. The sample size of these projects is adequate for training clustering and classification models.

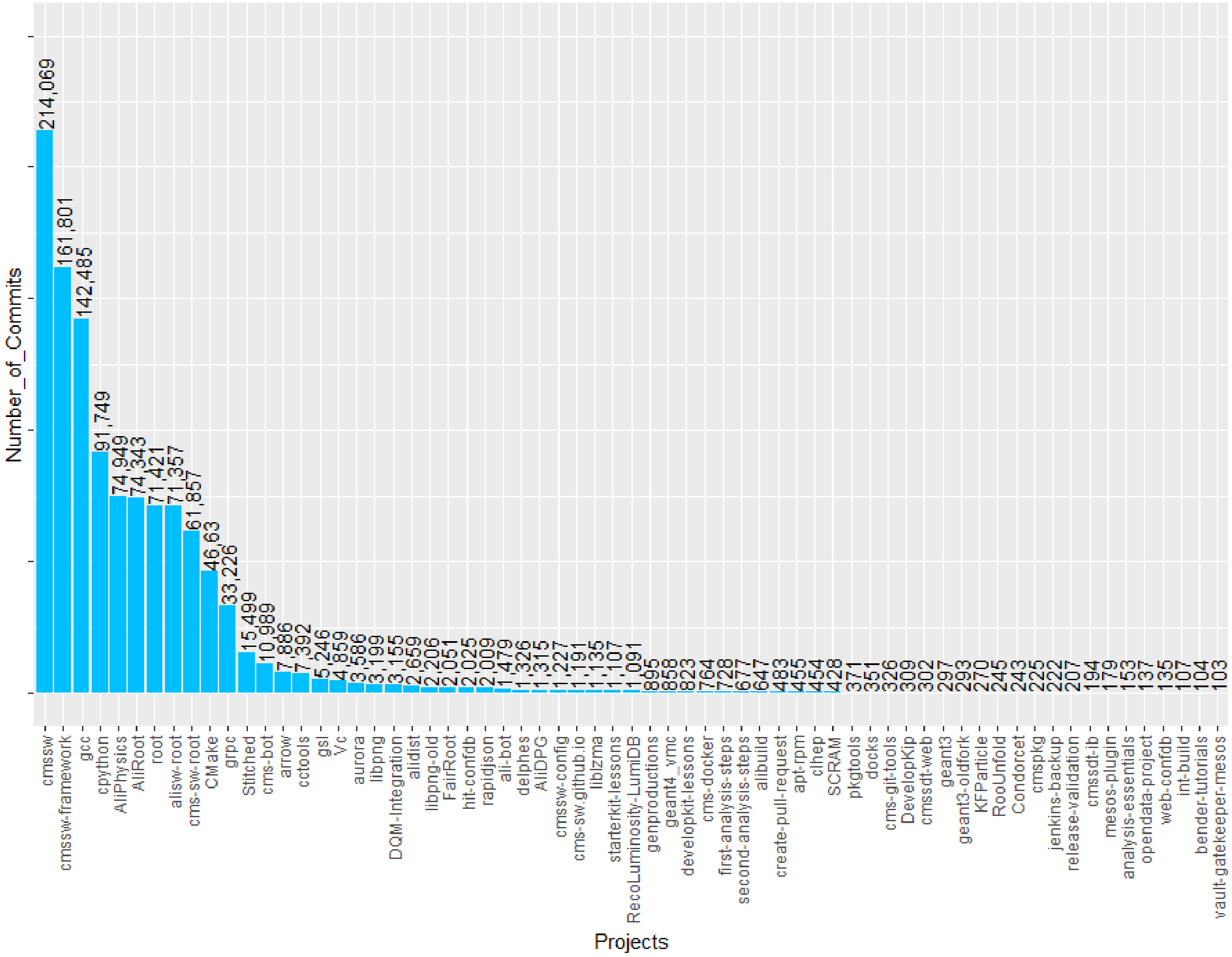


Figure 3. Number of commits per project.

Based on previous studies [31,41,58–60] and the knowledge of experts in software engineering, we have identified categories of types of the 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 analysed: we have especially widened 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 to keep track of general code change activities, while the *type of corrections* is mainly related to software problems.

Table 3. Types of changes and corrections.

Category	Values		
Type of changes	Development	Improvement	Performance
	Optimization	Cancellation	Documentation [31]
	Installation	Deployment	Debugging [31]
	Build [31]	Upgrade	Versioning [31]
	Refactoring [31]	Testing [31]	Feature Addition
	Update	Configuration	Dependency
	Indentation [31]	Initialization [31]	Platform [31]
	Corrective [31]	Branch [31]	Legal [31]
	Merge [31]	Revert	New Functionality
Type of corrections	Bug [31,60]	Issue	Solve
	Patch [59,60]	Abort [59,60]	Error [59,60]
	Bad [59,60]	Conflict [59,60]	Problem [59,60]
	Wrong [59,60]	Mistake	Avoid [59,60]
	Warning [59,60]	Fix	Anomaly
	Failure Exception	Crash	Incident
	Side Effect	Fail	Defect
	Glitch		

Table 4 shows an example of commits with two categories of types of commits. 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 corresponding 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

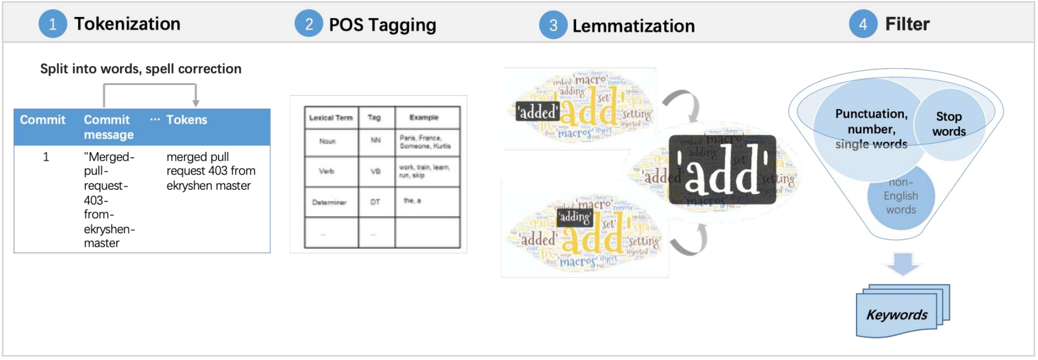
In this research, we have implemented an automatized labelling procedure based on the two variable *type of change* and *type of correction* with the aim of checking the categories obtained as a result of the ML classification techniques and using the resulting dataset for the ground-truth activity [61].

5. Data Preprocessing

Figure 4 shows in more detail the steps we have followed.

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, and 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 [21,62] 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.



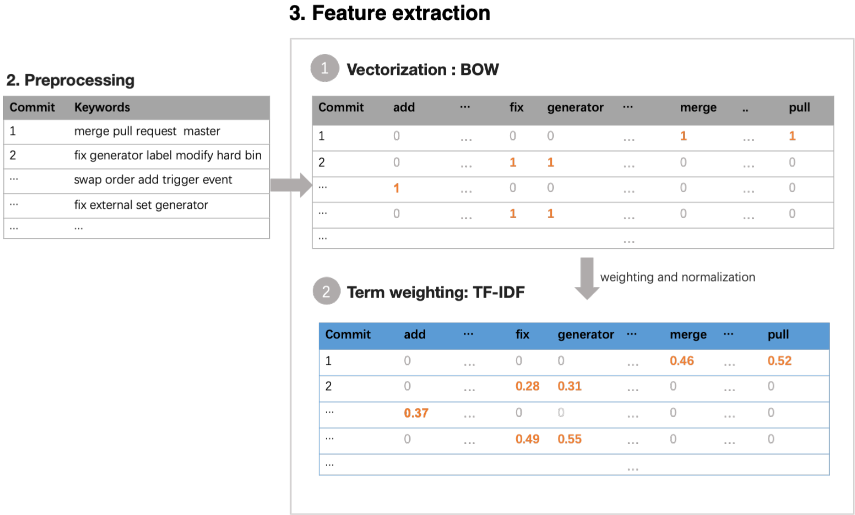
Commit	Commit message	···	Tokens
1	"Merged-pull-request-403-from-ekryshen-master		merged pull request 403 from ekryshen master

Lexical Term	Tag	Example
Noun	NN	Paris, France, Someone, Kurtis
Verb	VB	work, train, learn, run, skip
Determiner	DT	the, a
...	...	

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

6. Feature Extraction

Figure 5 shows more in detail how the process of feature extraction works. We have used BoW and TF-IDF techniques.



Commit	Keywords
1	merge pull request master
2	fix generator label modify hard bin
···	swap order add trigger event
···	fix external set generator
···	···

Commit	add	···	fix	generator	···	merge	..	pull
1	0	...	0	0	...	1	...	1
2	0	...	1	1	...	0	...	0
···	1	...	0	0	...	0	...	0
···	0	...	1	1	...	0	...	0
···					...			

Commit	add	···	fix	generator	···	merge	···	pull
1	0	...	0	0	...	0.46	...	0.52
2	0	...	0.28	0.31	...	0	...	0
···	0.37	...	0	0	...	0	...	0
···	0	...	0.49	0.55	...	0	...	0
···					...			

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

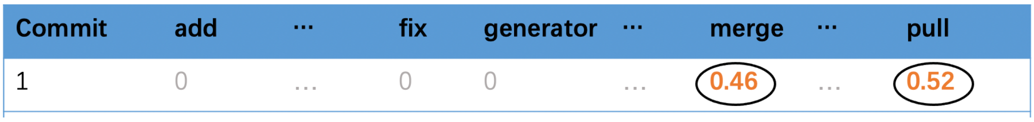
Equation (5) introduces the value of TF-IDF

$$TF\text{-}IDF = tf(t,d) \times idf(t) \tag{5}$$

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

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; and 3 projects have 15 to 16 clusters.

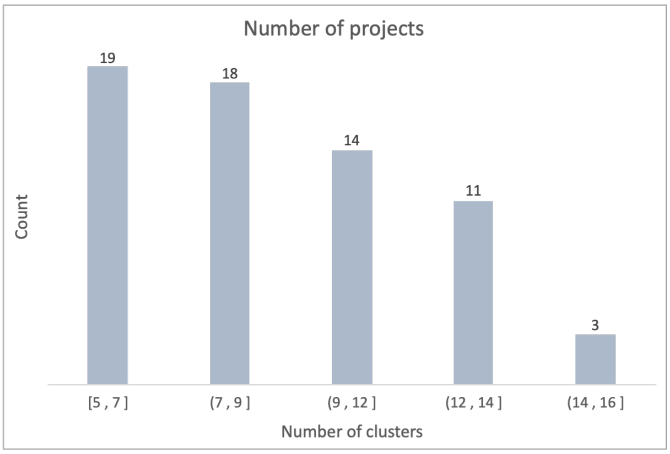


Figure 7. Number of clusters distribution of studied groups of projects (i.e., ALISW, LHCb, CMS-RW, and ROOT).

Figure 8 shows the AliDPG project (in the ALISW group) 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 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 Figure 8), 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.

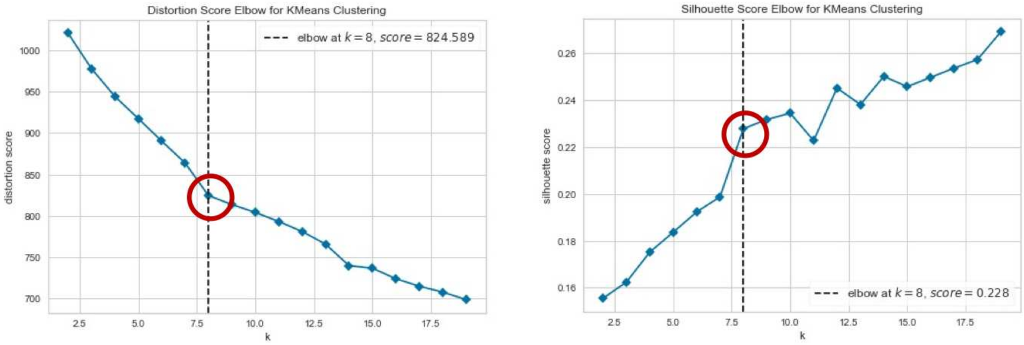


Figure 8. AliDPG—number of clusters. *k* values with the Elbow method on the left side and the silhouette coefficient score on the right side.

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 applied 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, and 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 analysed. 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 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—examples of 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, and merge. Cluster 7 only contains commits that have the change combination of branch and merge without any correction.

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 Third Party 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**

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 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 [51,63]. 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 [63]. 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):

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

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

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

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 and Validity

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

RQ1: What has been the impact of NLP techniques on the data preparation step? By applying NLP techniques in the data preparation phase (i.e., data preprocessing, feature extraction, and 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 belonging to HEP experiments, i.e., ALISW, LHCb, CMS-SW, and ROOT:

$$\begin{split} &\textit{feature reduction percentage per project} = (\textit{number of features before NLP} - \\ &\textit{number of features after NLP})/(\textit{number of features before NLP}) \end{split} \tag{9}$$

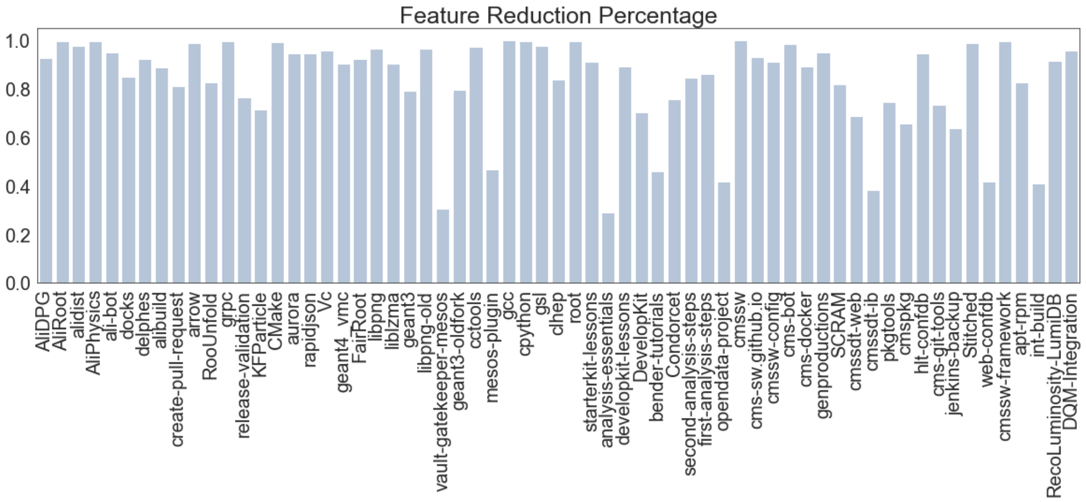


Figure 9. Feature reduction percentage for studied projects.

According to Figure 9, in AliDPG, for example, 93% features (all words in original commit messages) have been removed, only 7% features have been used in clustering and classification phases. NLP helps to remove unimportant and redundant words and only considers important keywords as features.

RQ2: Which ML classification techniques have the best performance of classifying new commits? As shown in Table 8, Random Forest (RF), Bagging (BG), and Decision Tree (DT) have the highest average metric scores (i.e., accuracy, recall, precision, and F1-score).

The best classifier of the considered methods is RF, which implies that it has outperformed other models for studied projects.

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**
Decision 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**

In addition, NLP techniques have an important impact on our results, because the F1-scores have improved. The average F1-score of all examined projects has reached 0.9360 (see Table 8 for RF) and the maximum F1-score has achieved 0.9496. In our previous study [18], without NLP techniques, the maximum F1-score was 0.8210 achieved by MNB.

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

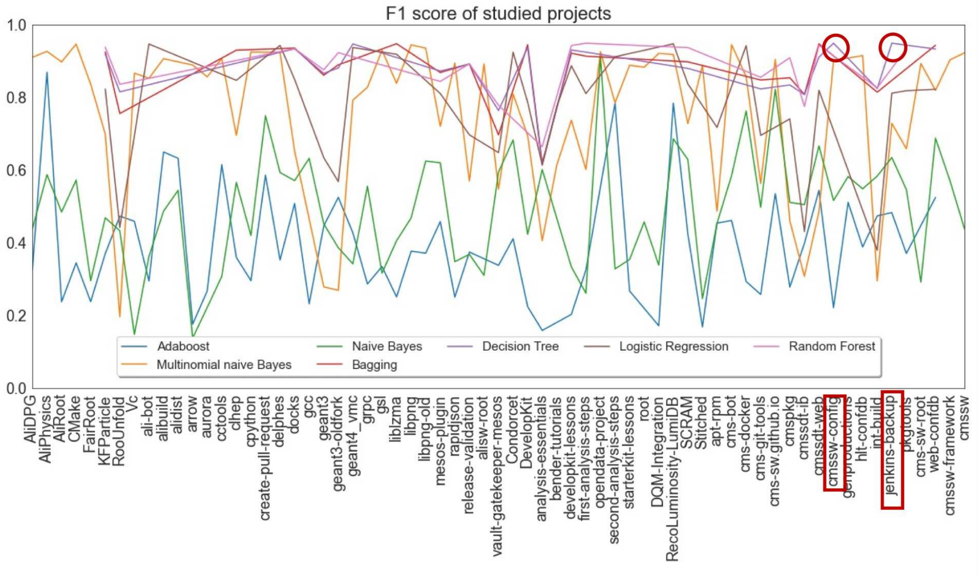


Figure 10. F1-score per project.

RQ3: Do the models behave differently for different projects? Table 9 summarizes the statistics of performance metrics of the ML methods adopted. AdaBoost (AB), Logistic Regression (LR), Multinomial Naive Bayes (MNB), and Naive Bayes (NB) methods show huge variations in performances over projects. On the contrary, Bagging, Decision Tree, and Random Forest show stability.

Table 9. The statistics of performance metrics.

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	0.973	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

In this study, the Coefficient of Variation (CV) of performance metrics 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 11–14 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.

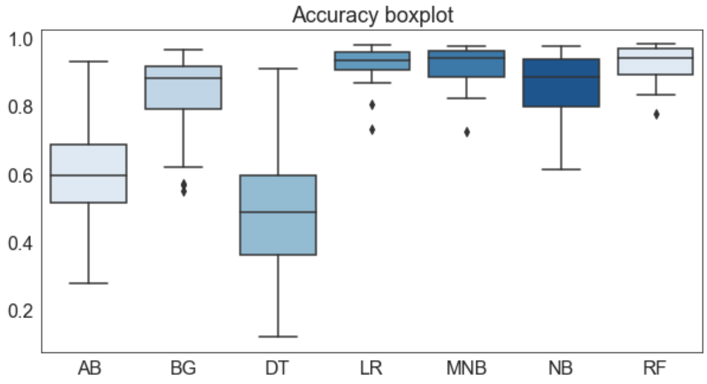


Figure 11. Accuracy boxplot.

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.

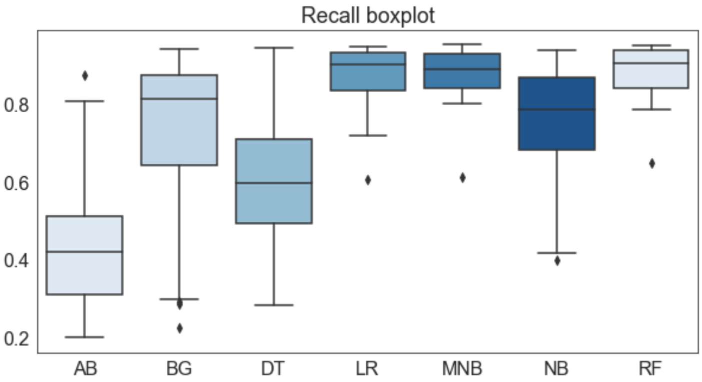


Figure 12. Recall boxplot.

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.

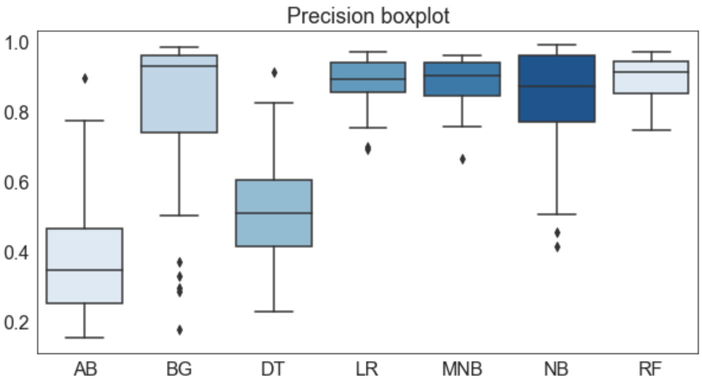


Figure 13. Precision boxplot.

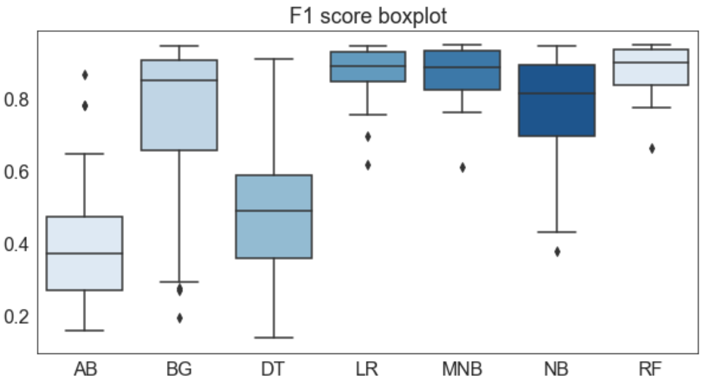


Figure 14. F1 score boxplot.

10. Conclusions

The study is motivated by the demand for analysing 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 have been well manipulated by NLP prepossessing tools and the 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 [18] 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.R. 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.

Funding: This research received no external funding.

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), Limassol, Cyprus, 31 August–2 September 2016; pp. 63–67. [CrossRef]
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, Genova, Italy, 5–8 March 2013; pp. 323–326. [CrossRef]
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), Coimbra, Portugal, 12–15 October 2020; pp. 37–42. [CrossRef]
4. Bavota, G. Mining unstructured data in software repositories: Current & future trends. In Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Osaka, Japan, 14–18 March 2016; pp. 1–12. [CrossRef]
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), Urbana, IL, USA, 30 October–3 November 2017; pp. 135–146. [CrossRef]
6. Jalote, P. *An Integrated Approach to Software Engineering*; Texts in Computer Science; Springer: Boston, MA, USA, 2005. [CrossRef]
7. Yalla, P.; Sharma, N. Integrating Natural Language Processing and Software Engineering. *Int. J. Softw. Eng. Its Appl.* **2015**, *9*, 127–136. [CrossRef]
8. Venigalla, A.S.M.; Chimalakonda, S. Understanding Emotions of Developer Community Towards Software Documentation, In Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), Madrid, Spain, 25–28 May 2021; pp. 87–91. [CrossRef]
9. Garousi, V.; Bauer, S.; Felderer, M. NLP-assisted software testing: A systematic mapping of the literature. *Inf. Softw. Technol.* **2020**, *126*, 106321. [CrossRef]
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), London, ON, Canada, 18–21 February 2020.

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. *arXiv* **2020**, arXiv: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 Trans. Softw. Eng.* **2016**, *42*, 379–402. [CrossRef]
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), Hamburg, Germany, 25–26 March 2019; pp. 238–241. [CrossRef]
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), Campobasso, Italy, 20 March 2018; pp. 25–30. [CrossRef]
15. Catolino, G.; Palomba, F.; De Lucia, A.; Ferrucci, F.; Zaidman, A. Enhancing change prediction models using developer-related factors. *J. Syst. Softw.* **2018**, *143*, 14–28. [CrossRef]
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 Trans. Softw. Eng.* **2009**, *35*, 607–623. [CrossRef]
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. [CrossRef]
18. 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 IEEE NSS MIC 2020, Boston, MA, USA, 31 October–7 November 2020.
19. Ronchieri, E.; Canaparo, M.; Yang, Y. Using Natural Language Processing to Extract Information from Unstructured code-change version control data: Lessons learned. In Proceedings of the International Symposium on Grids & Clouds, Taipei, Taiwan, 22–26 March 2021; p. 025. [CrossRef]
20. Piris, Y.; Gay, A.C. Customer satisfaction and natural language processing. *J. Bus. Res.* **2021**, *124*, 264–271. [CrossRef]
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, Trabzon, Turkey, 2–4 July 2012; pp. 1–5. [CrossRef]
22. 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), Ghaziabad, India, 22–23 February 2012; pp. 858–862.
23. dos Santos, G.E.; Figueiredo, E. Commit Classification using Natural Language Processing: Experiments over Labeled Datasets. In Proceedings of the CIbSE, Curitiba, Brazil, 9–13 November 2020.
24. Nyamawe, A.S. Mining commit messages to enhance software refactorings recommendation: A machine learning approach. *Mach. Learn. Appl.* **2022**, *9*, 100316. [CrossRef]
25. Sagar, P.S.; AlOmar, E.A.; Mkaouer, M.W.; Ouni, A.; Newman, C.D. Comparing Commit Messages and Source Code Metrics for the Prediction Refactoring Activities. *Algorithms* **2021**, *14*, 289. [CrossRef]
26. Rebai, S.; Kessentini, M.; Alizadeh, V.; Sghaier, O.B.; Kazman, R. Recommending refactorings via commit message analysis. *Inf. Softw. Technol.* **2020**, *126*, 106332. [CrossRef]
27. Rantala, L.; Mäntylä, M. Predicting technical debt from commit contents: Reproduction and extension with automated feature selection. *Softw. Qual. J.* **2020**, *28*, 1–29. [CrossRef]
28. Jung, T. CommitBERT: Commit Message Generation Using Pre-Trained Programming Language Model. *arXiv* **2021**, arXiv:2105.14242.
29. Yan, M.; Zhang, X.; Liu, C.; Xu, L.; Yang, M.; Yang, D. Automated change-prone class prediction on unlabeled dataset using unsupervised method. *Inf. Softw. Technol.* **2017**, *92*, 1–16. [CrossRef]
30. Levin, S.; Aviv, T.; Aviv, T. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. *arXiv* **2017**, arXiv:1711.05340.
31. Messaoud, M.B. On the Classification of Software Change Messages using Multi-label Active Learning. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019. [CrossRef]
32. 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. In Proceedings of the 13th Working Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, 14–22 May 2016; pp. 496–499. [CrossRef]
33. Levin, S.; Yehudai, A. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. In Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, Toronto, ON, Canada, 8 November 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 97–106. [CrossRef]
34. 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, Tampa, FL, USA, 25–26 March 2004. [CrossRef]
35. Hattori, L.P.; Lanza, M. On the nature of the nature of law. *Arch. Rechts Sozialphilosophie* **2012**, *98*, 457–467. [CrossRef]
36. Yamauchi, K.; Yang, J.; Hotta, K.; Higo, Y.; Kusumoto, S. Clustering commits for understanding the intents of implementation. In Proceedings of the 30th International Conference on Software Maintenance and Evolution, ICSME 2014, Victoria, BC, Canada, 29 September–3 October 2014; pp. 406–410. [CrossRef]
37. Zhang, Z.; Guo, J.; Zhang, H.; Zhou, L.; Wang, M. Product selection based on sentiment analysis of online reviews: An intuitionistic fuzzy TODIM method. *Complex Intell. Syst.* **2022**, *8*, 3349–3362. [CrossRef]

38. Zhou, L.; Tang, L.; Zhang, Z. Extracting and ranking product features in consumer reviews based on evidence theory. *J. Ambient. Intell. Humaniz. Comput.* **2022**, 1868–5145. [CrossRef]
39. Zhou, L.; Zhang, Z.; Zhao, L.; Yang, P. Attention-based BiLSTM models for personality recognition from user-generated content. *Inf. Sci.* **2022**, *596*, 460–471. [CrossRef]
40. Zhang, H.; Zhang, Z. Characteristic Analysis of Judgment Debtors Based on Hesitant Fuzzy Linguistic Clustering Method. *IEEE Access* **2021**, *9*, 119147–119157. [CrossRef]
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. *Math. Probl. Eng.* **2021**, *2021*, 1–30. [CrossRef]
42. 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), Bahawalpur, Pakistan, 5–7 November 2020; pp. 1–6. [CrossRef]
43. 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), Waknaghat, India, 22–24 December 2016; pp. 202–206. [CrossRef]
44. Alsarhan, Q.; Ahmed, B.S.; Bures, M.; Zamli, K.Z. Software Module Clustering: An In-Depth Literature Analysis. *IEEE Trans. Softw. Eng.* **2020** *48*, 1905–1928. [CrossRef]
45. 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. *J. Phys. Conf. Ser.* **2019**, *1361*. [CrossRef]
46. Yuan, C.; Yang, H. Research on K-Value Selection Method of K-Means Clustering Algorithm. *J* **2019**, *2*, 226–235. [CrossRef]
47. Kaoungku, N.; Suksut, K.; Chanklan, R.; Kerdprasop, K.; Kerdprasop, N. The silhouette width criterion for clustering and association mining to select image features. *Int. J. Mach. Learn. Comput.* **2018**, *8*, 69–73. [CrossRef]
48. Berkhin, P. A Survey of Clustering Data Mining Techniques. In *Grouping Multidimensional Data: Recent Advances in Clustering*; Kogan, J., Nicholas, C., Teboulle, M., Eds.; Springer Berlin/Heidelberg, Germany; New York, NY, USA, 2006; pp. 25–71. [CrossRef]
49. Daniel Jurafsky, J.H.M. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*; PEARSON: London, UK, 2008 ; pp. 1–306. [CrossRef]
50. Dönmez, P. Introduction to Machine Learning. *Nat. Lang. Eng.* **2013**, *19*, 285–288. [CrossRef]
51. Sutton, C.D. Classification and Regression Trees, Bagging, and Boosting. In *Data Mining and Data Visualization*; Rao, C., Wegman, E., Solka, J., Eds.; Elsevier: Amsterdam, The Netherlands, 2005; Volume 24, pp. 303–329. [CrossRef]
52. Browne, M.W. Cross-Validation Methods. *J. Math. Psychol.* **2000**, *44*, 108–132. [CrossRef]
53. 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. *Comput. Methods Programs Biomed.* **2020**, *196*, 105604. [CrossRef]
54. ALISW. 2022. Available online: https://github.com/alisw (accessed on 15 September 2022).
55. LHCB. 2022. Available online: https://github.com/lhcb (accessed on 15 September 2022).
56. CMS-SW. 2022. Available online: https://github.com/cms-sw (accessed on 15 September 2022).
57. ROOT. 2022. Available online: https://github.com/root-project/root (accessed on 15 September 2022).
58. Swanson, E.B. The dimensions of maintenance. In Proceedings of the 2nd International Conference on Software Engineering, San Francisco, CA, USA, 13–15 October 1976; pp. 492–497.
59. Hindle, A.; German, D.M.; Holt, R. What Do Large Commits Tell Us? A Taxonomical Study of Large Commits. In Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR '08, Leipzig, Germany, 10–11 May 2018; Association for Computing Machinery: New York, NY, USA, 2008; pp. 99–108. [CrossRef]
60. Islam, K.; Ahmed, T.; Shahriyar, R.; Iqbal, A.; Uddin, G. Early prediction for merged vs abandoned code changes in modern code reviews. *Inf. Softw. Technol.* **2022**, *142*, 106756. [CrossRef]
61. Golzadeh, M.; Decan, A.; Legay, D.; Mens, T. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *J. Syst. Softw.* **2021**, *175*, 110911. [CrossRef]
62. 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), Austin, TX, USA, 16–17 May 2016; pp. 1–3. [CrossRef]
63. Lan, M.; Tan, C.L.; Su, J.; Lu, Y. Supervised and Traditional Term Weighting Methods for Automatic Text Categorization. *IEEE Trans. Pattern Anal. Mach. Intell.* **2009**, *31*, 721–735. [CrossRef]