Using Natural Language Processing to Extract Information from Unstructured code-change version control data: lessons learned

E. Ronchieri [1] M. Canaparo [1] Y. Yang [2]
D. Salomoni [1] D. C. Duma [1] A. Costantini [1]

[1]INFN CNAF, Bologna Italy

[2]Department of Statistical Sciences, University of Bologna, Italy

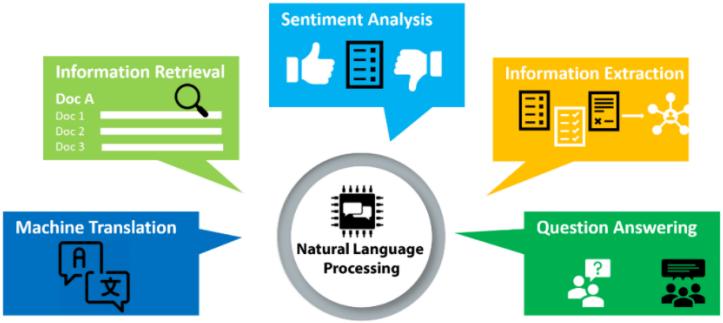
International Symposium on Grids & Clouds
March 21-26, 2021

Outline

Table of Contents

Natural Language Processing (NLP)

NLP is an area of artificial intelligence that uses computer science to elaborate on human languages [1, 2].



source `https://www.ontotext.com/blog/top-5-semantic-technology-trends-2017/`

NLP and Code Changes

Many **NLP** techniques have been **employed** to **extract key information** from free-form text or to **generate models** from the analysis of text. They have been **also** applied to **categorize code changes** according to their commit messages.
Why applying **NLP** to Software Engineering tasks?

- the detection of developers' emotions;
- the detection of the opinions of a software product;
- the improvement of test case selection;
- the detection of error-prone software components.

Some **NLP**-based approaches have focused on **code changes** and their consequences in terms of, for instance, code review and mapping of bug reports to relevant files. [3]

Code Changes

Code changes:

- are typically introduced for new features, bug fixing, code refactoring,
- they can occur during both the development and maintainance phase.

They may be the cause of the introduction of new defects.
Version control and source code management services (such as GitHub) store all the history of **code changes** thus:

- can be used to identify change-prone software modules;
- provide some insights about how software has been developed and maintained, leading to
  - limitation in software defect-proneness;
  - improvement in software maintainability.

Code Change Prediction Model

- **Predicting the areas of code having a higher likelyhood to change** is essential to help developers for maintenance operations.
- **Machile Learning** (ML) techniques have been **largely exploited** in this scenario.
- **ML-based models** relate a set of indipendent variables **extracted from code repositories** to a dependent variable (**change-proneness of a module**).
- **Previous ML approaches** in literature showed their general **effectiveness**, however, they might produce different results according to the different ML techniques. [4]
- In our work, we have **employed ML** techniques both for **clustering** and **classification** tasks.

Why this study?

- Software changes (e.g. in GitHub) are annoted by **commit messages**.
- Developers use commit messages **to keep track of the changes** performed during the development process.
- An accurate **classification** of these **messages** can contribute to help monitor the evolution of a software project.
- However, **analysis of commit messages** is **challenging** because of:
  - their **inconsistency** with the project updates [5],
    - ⋆ more than one updates in the same commit message;
  - the use of **informal language**.
- Furthermore, previous approaches in literature assumed that a **commit message** could be associated with **one label at a time**; on the contrary, developers often perform **more tasks in a single commit**. [6]

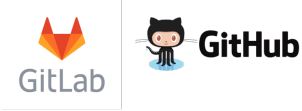
Why this study?

- In this study we aim at:
  - building a **historical code change dataset** composed of **commit messages** classified according to their types of changes and corrections;
  - enabling **systematic queries** to the **structured data** to show the evolution of a software project,
  - following a **multi-label classification approach**, where a commit message can be associated with more than one class label.
- Our final goal is to provide developers - in the HEP context - with a **tool for the automatic classification of commit messages**.
- This **tool** would be helpful to monitor the development and maintenance activities at any point in time and to improve software development practice.

Table of Contents

Experimental tools

Tools to collect **commit messages** and extract **code changes**:



Tools to elaborate and relate **heterogeneous data**

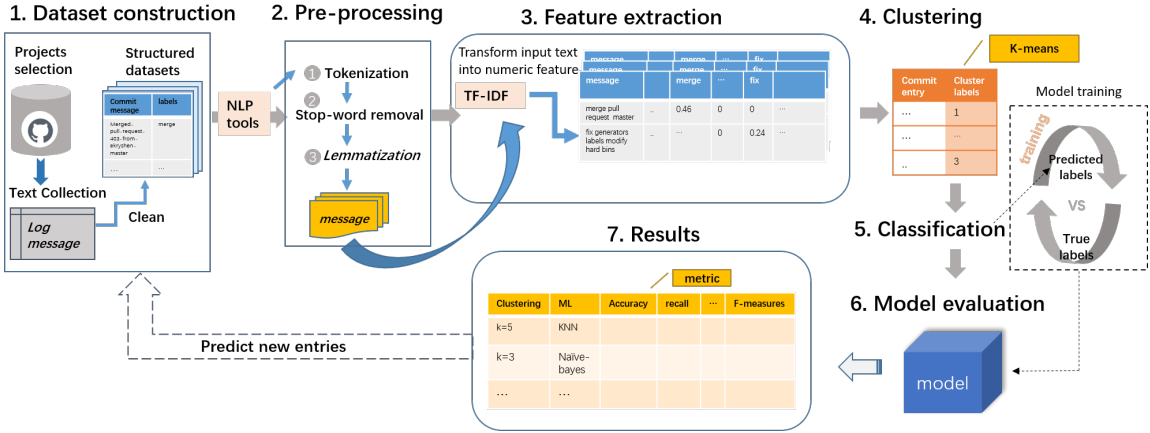
Examined Projects

ALISW: 102 projects	**LHCB**: 22 projects	**CMS-SW**:41 projects	**ROOT**: 1 project	**GEANT4**: 1 project
AliDPG	starterkit-lessons	cms-prs	root	geant4
AliRoot	starterkit-ci	cmssw		
alidist	glossary	cms-sw.github.io		
AliPhysics	analysis-essentials	cmssw-config		
AliGenerators	starterkit	cmssw-cfipython		
ali-bot	developkit-lessons	cms-bot		
docks	ostap-tutorials	cms-docker		
dim	DevelopKit	genproductions		
delphes	gitbook-plugin-panels	SCRAM		
alibuild	starterkit-receipts	root		
pythia6	bender-tutorials	cmssdt-web		
homebrew-system-deps	Condorcet	cmssdt-wiki		
create-pull-request	second-analysis-steps	cmssw-wm-tools		
arrow	first-analysis-steps	cmssdt-ib		
RooUnfold	opendata-project	pkgtools		
alisw.github.io	plugin-codesnippet	cmspkg		

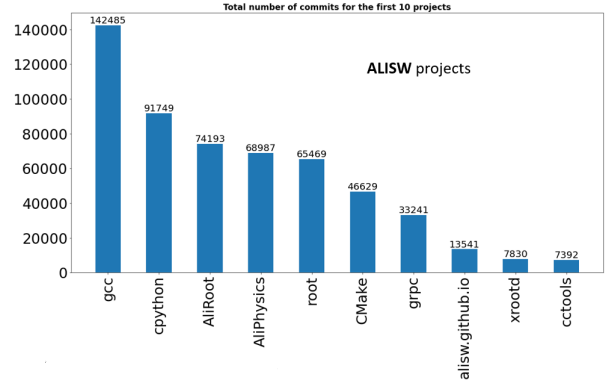
Number of projects: 167

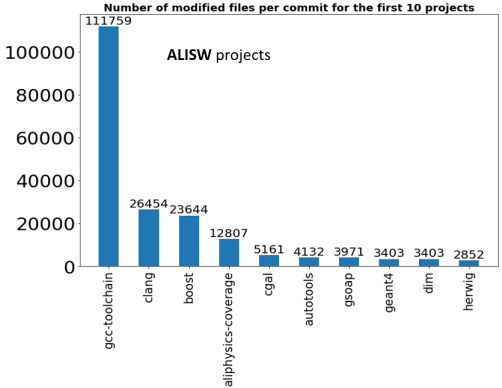
```
https://github.com/alisw
https://github.com/cms-sw
https://github.com/lhcb
https://github.com/root-project/root
https://github.com/Geant4/geant4
```

Methodology



Dataset Construction





Collected up in Nov 2020.
Log Statistics:

Total number of days	Total number of authors
Total number of commits	**Total number of modified files**
Number of modified files per commit	Number of authors per commit
Start date	Period of time [start date, end date]

Dataset Construction

Code changes key terms from log messages

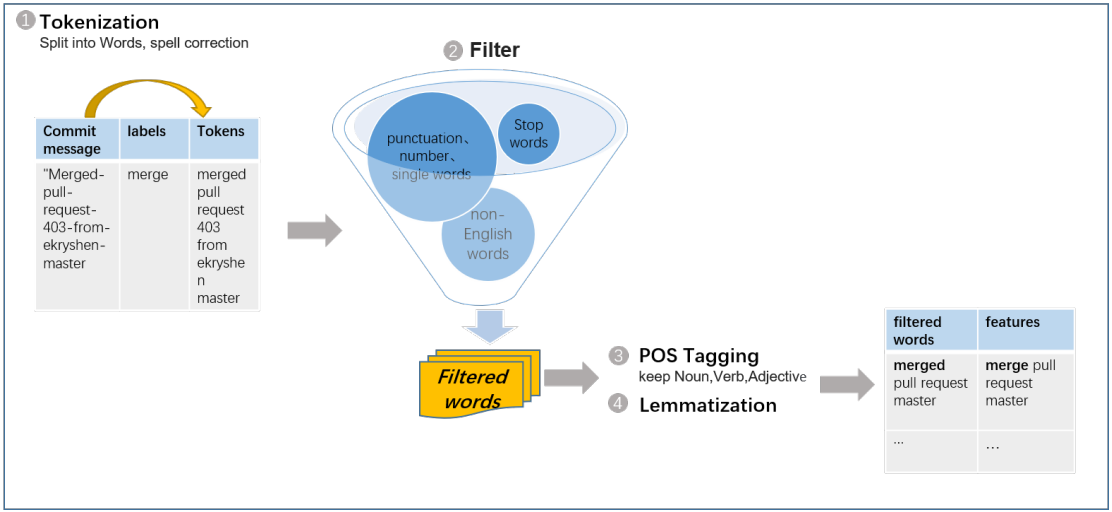
Development	Improvement	Performance	Optimization	Cancellation
Documentation	Installation	Deployment	**Debugging**	**Build**
Upgrade	**Versioning**	**Refactoring**	New Functionality	**Testing**
Feature Addition	Update	Configuration	Dependency	**Indentation**
Initialization	**Platform**	**Corrective**	**Branch**	**Legal**
Merge	Revert			

Fix	Anomaly
Bug	Failure
Issue	Error
Solve	Exception
Patch	Crash
Abort	Glitch
Bad	Defect
Conflict	Incident
Problem	Side Effect
Wrong	Fail
Mistake	

- Terms in bold cited in [7]

Pre-processing - NLP

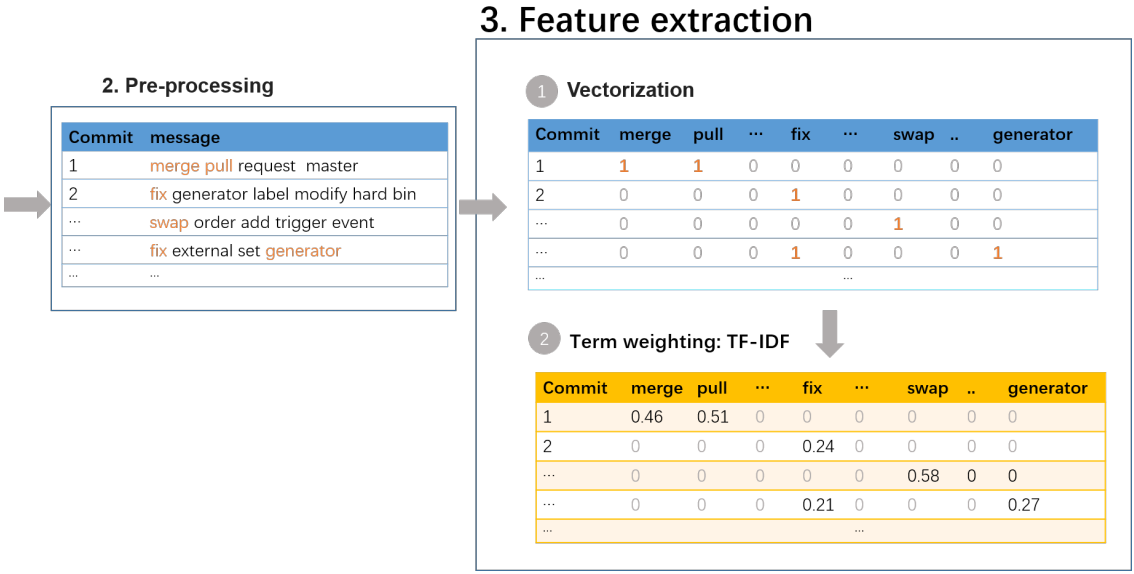
2. Pre-processing



Pre-processing

After the Lemmatization process

Feature Extraction



Feature Extraction

TF-IDF [7, 8] is an information retrieval technique used to evaluate how relevant a word (i.e. term) is to a document (**commit message**) in a collection of documents (**log messages**).
TF-IDF = $tf(t, d) \times idf(t)$ uses two metrics:

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

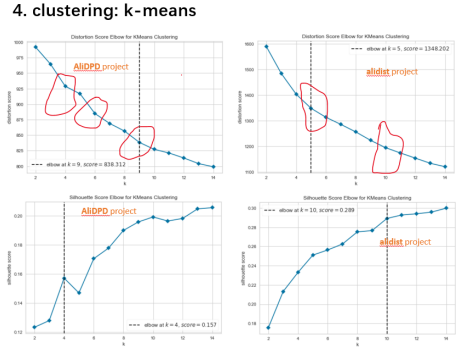
The higher the TF-IDF score, the more relevant that term is in that particular document.

2 **Term weighting: TF-IDF**

Commit	merge	pull	···	fix	···	swap	..	generator
1	0.46	0.51	0	0	0	0	0	0

Clustering and Classification

- Used the **k-means** clustering method on feature matrix to identify clusters of commit messages.



- Applied **the Elbow method** and **Silhouette score** to determine the number of clusters **k**.
- The Elbow method is more of a decision rule, while the Silhouette is a metric used for validation while clustering. Thus, it can be used in **combination** with the Elbow method for a more confident decision.

ML techniques application

- **Assessment** of whole **code change database**, composed of one dataset per project, **through** the application of **ML** techniques.
- Reserved 30% of the dataset for validation and the remaining for training of the model.
- Comparison of the different ML techniques according to some performance metrics (e.g. F-measure).
  - F-measure=$2 \times \frac{(precision \times recall)}{(precision + recall)}$
  - It determines the entries (e.g. modules or files associated to commit messages) that were predicted as belonging to a particular cluster (a combination of types of changes and corrections).
- ML algorithms used NaiveBayes, NaiveBayesMultinomial, RandomForest, LogisticRegression, AdaBoost, DecisionTree, Bagging.

Table of Contents

Answering Research Questions

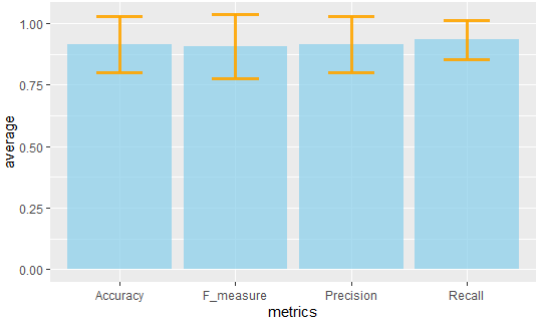
- RQ1: What has been the impact on NLP techniques for the construction of our dictionary and the feature definition?
  - NLP techniques proved to have an important impact on our work.
  - The number of features decreased.
  - In our previous study [9] without NLP, the maximum value of F-measure was 0.8210.

Project Name	#features without NLP	#features with NLP	Decreased percentage
1	1660	351	78,86%
2	974	411	57,80%
3	2487	483	80.58%
4	500	160	68.00%
5	1382	321	76.77%
6	405	177	56.30%

More results will be included in the proceedings.

Answering Research Questions

- RQ2: Is the constructed database suitable for code-change prediction? Based on the F-measure our analysis shows:
  - the database is a suitable input for code-change predition models that rely on ML techniques;
  - F-measure values reach values up to 0.9389 that is a promising result.



Answering Research Questions

- RQ3: Which ML algorithms or algorithm family perform the best in code-change prediction?
  - Decision Tree, Bagging, Logistic Regression and Random Forest have the highest F-measure values in the majority of the examined projects.

Threats to Validity

The **major threats to validity** related to our research are:

- **projects' bias**: we have considered only HEP projects stored in GitHub;
- **programming language bias**: the majority of software projects are written in C++, this may have affected commit messages.
- **data labelling**: in general, there is no agreement on how commit messages should be labeled.

However, to **overcome** these limitations we have relied on **peer-reviewing** and a **partial automation of the process**.

Table of Contents

Conclusions

- We have experienced that **software development activities** are widely **detailed** in **software repositories**.
- We have built a **historical code change dataset** of **categorized commit messages** by following a **multi-label classification** approach.
- We have provided a **dictionary** for **labeling commit messages** based on labels both derived from existing literature and found in the analysed commit messages.
- We believe our work may help **monitoring** and **improving** the **development and maintenance processes**.

Future Work

- In the future, we plan to:
  - **analyze misclassified commits**;
  - include **other commit messages** that might not be available in the current dataset;
  - **incorporate**, if possible, the **developers' community feedback** to define more suitable code change category;
  - finalize our **tool** for the **automatic classification of commit messages**.

Thanks for your attention
Any questions?

- Email addresses
  - **Elisabetta Ronchieri, elisabetta.ronchieri@cnaf.infn.it**
  - Marco Canaparo, marco.canaparo@cnaf.infn.it
  - Yue Yang, yue.yang3@studio.unibo.it
  - Davide Salomoni, davide.salomoni@cnaf.infn.it
  - Alessandro Costantini, alessandro.costantini@cnaf.infn.it
  - Doina Cristina Duma, cristina.aiftimiei@cnaf.infn.it

Table of Contents

References

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

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

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

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

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

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

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

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

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