On the Classification of Software Change Messages using Multi-label Active Learning

Sirine Gharbi
LARODEC - Institut Supérieur de Gestion de Tunis
41, Avenue de la liberté, 2000, Le Bardo, Tunisia
gharbi.cyrine@outlook.com

Mohamed Wiem Mkaouer
Rochester Institute of Technology
Rochester, NY, USA
mwmvse@rit.edu

Ilyes Jenhani
Prince Mohammad bin Fahd University
Khobar, Saudi Arabia
ijenhani@pmu.edu.sa

Montassar Ben Messaoud
LARODEC - Institut Supérieur de Gestion de Tunis
41, Avenue de la liberté, 2000, Le Bardo, Tunisia
montassar.benmassaoud@isgs.u-sousse.tn

ABSTRACT

In this paper, we present a multi-label active learning-based approach to handle the problem of classification of commit messages. The approach will help developers track software changes, e.g., adding or updating existing features, fixing user-reported errors, improving software performance, etc. We first constructed an unlabeled dataset of commit messages where each commit message is represented as a vector of feature values. The set of adopted features were automatically generated from the original commit messages using Term Frequency-Inverse Document Frequency (TF-IDF) technique. Because many commit messages can be assigned more than one commit class at the same time and in order to reduce the effort needed to assign labels to each instance in a large set of commit messages, we adopted an Active Learning multi-label approach. Experimentations have shown that we could train an accurate multi-label classifier model, in our case, a binary relevance with logistic regression as a base classifier, by actively querying an oracle for labels during the training process and with a reasonable number of labeled instances.

CCS CONCEPTS

• **Theory of computation** → **Active learning**; • **Software and its engineering** → ***Software creation and management***; • **Software post-development issues** → **Software evolution** ;

KEYWORDS

software maintenance, commit categorization, natural language processing

1 INTRODUCTION

Collaborative and social-coding platforms have become the backbone of open source and industrial software development. They organize the distributed and concurrent implementation and maintenance of software, when performed by several programmers. The transparent nature of these platforms allows the exploration and analysis of previously executed code changes, for various reasons related to software engineering tasks such as, root cause analysis, bug localization, objects traceability, approximation of development effort and cost, etc. Each code change, i.e., commit, is characterized by a message, written in natural language by the committer, describing the intention behind the committed change.

Commit messages are the main source for developers to track the software evolution in terms of adding or updating existing features, fixing user-reported errors , improving software performance, etc. However, the growth of the number and variety of development activities, makes their documentation, through commit messages, non-trivial. Therefore, the manual exploration of commits is challenging for developers. Furthermore, the subjective nature of text descriptions may hinder their understanding, especially when developers belong to various backgrounds and don't share common naming conventions [16]. Thus, the manual classification is a subjective, context-biased and human-intensive task.

In this context, several studies have been focusing on automatically classifying software changes based on the textual description of their commits messages. These approaches rely on Information Retrieval (IR) and Natural Language Processing (NLP) techniques to extract relevant features, from the commit text, which decipher the engineering nature of the code change and help developers in identifying relevant changes [19]. On the other hand, training the classifiers is expensive since it requires the manual classification of an initial set of commits, and so, using a limited set of commits challenges the scalability of the classification. According to Hindle et al. [16] classifiers average accuracy of 60%, when trained using a single project commits, decreases to an average of 52% when evaluated across projects. Moreover, many studies addressed the classification as a multi-class mono-label problem. Since developers typically interleave multiple engineering tasks in a single commit e.g., refactoring the design while adding a new feature, framing it in one class is not optimal and it deteriorates the right distribution of engineering tasks during the classification.

To cope with the above-mentioned limitations, we address the commit classification as a Multi-Label Active Learning problem. The active learning component decreases the necessary learning effort and allows the expansion of the training set across projects. The multi-labeling allows each commit to belong to more than one

SAC '19, April 8–12, 2019, Limassol, Cyprus
© 2019 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus*, https://doi.org/10.1145/3297280.3297452.

class and so, it helps in accurately capturing the type of engineering activity captured from the commit text.

We got our classification model by working on commits taken from 12 distinct open source projects from Github. So in this new approach we started our work by cleaning the data set, and preprocess the developer's comments, in addition, we performed the feature selection. Once we have our features which represent in our work the most relevant keywords, we label them according to the Swanson's maintenance activities [25].

The experimental results show a decreasing pace of the Hamming Loss (HL) [26] figure which corresponds to the set of the labels that are incorrectly predicted to the total number of labels and an increasing one of the F1 score that returns the harmonic mean of precision and recall, with an average HL value 0.05 and an average F1 score value 45.79%.

To summarize, our main contributions are as follows :

1. To the best of our knowledge, this is the first work to propose a Multi-Label Active Learning Approach to actively learn commit classification, to overcome the limitations of the multi-class classification.

2. The usage of the active learning classification in order to avoid labeling the whole dataset, while maintaining good classification outcomes.

2 BACKGROUND

In this section we present the necessary taxonomy related to commit messages and preliminaries on multi-label classification, which include an elaboration of NLP techniques, multi-label classification and multi-label active learning.

2.1 Commits

Version Control Systems (VCS) allow the management of code changes, applied throughout the lifecycle of software systems. It serves as depot for the history of each code update characteristics, including the committer identifier, commit time, ID, etc. Those changes are saved as commits which actually correspond to a subversion of a project branch or repository. So that the developers can navigate at any time those commits to check the cleanliness of the code as well as the operational update. Therefore, those commits are considered as checkpoints that illustrate the history of changes and allow restore the points through which the developer can view a version of a branch at a specific time or date. Moreover, commits are generally accompanied by messages, the latter describes the update done and it gives the monitors and supervisors transparency to understand what is happening in the workflow of the concerned repository.

As mentioned above, commits are considered as the software changes that might be as simple as fixing a User Interface error which usually requires minor changes or difficult such as adding a whole new feature from scratch that sometimes needs the removal or addition of classes or packages [8]. Recognizing and comprehending the nature of changes in a specific commit is worthy to developers, tester, and managers due to their implication into the software development.

For example, if a commit has been identified as a change in a class, then that latter must be retested, and additional tests must be integrated and implemented. Also, this change allows the developer to know that all the code elements that use this class will be impacted. In addition, the manager will have the opportunity to assess the costs and risks associated with different phases of deployment [8].

Also, the developers are suffering from constant turnovers, and the presence of the commits might be a very helpful way into the integration of the new developers, it allows them to have at least an idea about the work that has been done by the previous ones. So due to the variety, diversity and importance of commits, we decided in this work to use the 3 main categories proposed by [25] and explained in Table1.

Table 1: The commit categories

Corrective	In this category we deal with the correction of software bugs and defects especially those related to the user interface
Perfective	It concerns the improvement of the software design and the enhancement of the performance, also the correction of the source code
Adaptive	This class corresponds to the system modification in order to adapt it to the new environment such as the feature addition to making it more effective

2.2 Natural Language Processing

For grammatical reasons, documents use a different form of a word, that is why to select features (i.e. keywords) we preprocessed the commits in 3 steps : Stop-word removal, Stemming and Lemmatization.

2.2.1 Stop-Word Removal.

In general terms, Stop-word removal can be defined as an information retrieval to improve the algorithm results and is considered as a way to remove commonly used words that carry little meaning. These words include *is*, *are*, *am*, *if*, etc.

In our case we have two options whether we set a customized list of English stop-words or simply choose the English stop words as a criterion so that we can have relevant information while doing the feature selection.

2.2.2 Stemming.

The term stemming is generally understood to mean a process of reducing a word to its root form. For example, the word *writes* and *writing* would both be reduced to *write*.

As any other technique the stemming has issues when it comes to have relevant results, so while trying it, we noticed that the stemmer always removes the last letter for example *add-ad*, *remove-remov*. For this aim, we had to use the lemmatization technique to improve the results.

2.2.3 Lemmatization.

Lemmatization normally aims to remove inflectional endings only and return the base or dictionary form of a word, which is known as the lemma.

For our work, we used the WordNetLemmatizer which returns the input word unchanged if it cannot be found in WordNet, this allowed us to have better and significant keywords.

2.3 Multi-Label Classification

It is well known that the multi-class classification acquires an input to be associated with a one class label, yet sometimes these inputs need a flexible setting which enables them to be associated with a multi-class label and this is the case of many real-word problems such as classifying commits into the 3 maintenance activities where we can find into a single commit more than one task for example the developer added a new feature and fixed a bug so here we label this commit as corrective and adaptive.

So, multi-label classification is recognized as being crucial in text categorization [9], in sentiment analysis [27], and image classification [15]. As a result of such wide range of applications, in recent years, multi-label classification has become an emerging research area.

2.4 Multi-Label Active Learning

Labeling a multi-label commit is challenging, since it is time consuming and costly so, in this case, we need to use Active learning which is, according to [15], a mechanism that aims to optimize the classification performance while minimizing the number of needed labeled data for training. There are mainly three active learning approaches [5] :

— Membership query synthesis ;
— Stream-based selective sampling ;
— Pool-based sampling.

In the first approach the learner generates an instance. In the second one, data points are made available continuously in a stream-like fashion, and therefore decisions about whether an unlabeled instance should or not be labeled are made individually or in small batches [7]. The third setting which is the scenario that we adapted to our work, assumes that a pool of unlabeled data is made available from the onset of training [7].

3 PROBLEM STATEMENT

During the examination of commit messages for different applications, we noticed that many of these messages could not be uniquely classified as "Corrective" OR "Perfective" OR "Adaptive". In fact, if we look at the following example which is taken from our dataset :

Table 2: A commit message belonging to more than one class

App	Commit
a2dpvolume	Created a "car dock" device which has all the features of a normal bluetooth device. Fixed database error where on create it did not create all columns. Fixed the reload list issue from the edit device screen. I had not implemented the reload intent call.

The commit in Table 2 illustrates an example of code changes which corresponds to adding a container for the features of a bluetooth device while fixing errors related to information storage in the application's database. This commit message describes two maintenance categories, hence should be classified as : *perfective* and *corrective* at the same time. With multi-class classification approaches, where each instance must be labeled with one and only one class label, a solution to this problem is to duplicate the instance and assign to each copy one of the two class labels. The problem with this solution is that it loses some useful information about the dependency between these two (or more) class labels, which means dependencies between maintenance categories will be hard to detect. In this paper, we will adopt a multi-label classification approach where each instance can be assigned any subset of the set of all possible class labels. So, in a the multi-label setting, the above message will be assigned the set of labels : {perfective, corrective}.

Besides their multi-label nature, software repositories contain a huge number of commit messages making the labeling process of each one of these messages very hard, error-prone and time-consuming. Hence, instead of labeling the whole dataset we have (which contains almost 30000 unlabeled instances), we will adopt a multi-label Active Learning approach which aims to optimize the classification performance while minimizing the number of needed labeled training instances.

4 RELATED WORK

Mining commits has been widely adopted by researchers for addressing various problems that can be related to (re)assigning developers for bug and issues [1], extracting developers decisions [2] and eventually code change classification [3, 16, 22]. In this section, we firstly report studies related to the classification of commits messages with respect to the Swanson's model [25], then we enumerate the work related to active learning for multi-label classification.

4.1 Commit classification

Diverse studies have been performed on developer's comments in order to extract relevant data such as keywords which were indicative information for researchers to classify commits [10, 11, 13, 16, 22] in the scope of single projects and cross projects. In this alternative, Mockus et al. [12] used the word frequency analysis and normalization to select relevant keywords in order to do the classification. Levin et al. [22] used methods which are similar to Fisher et al. [13] to classify commits into the three categories and searched for relevant keywords by performing some techniques like stemming and case-folding. Other studies added auxiliary information related to authors [16] to classify developer's commits. Amor et al. [11] introduced a methodology that uses a slightly different technique without referring to the keywords using a Naive Bayes Classifier to obtain a fine-grained classification of code transactions from their textual description.

Recently, Levin et al. [23] suggested a method for automatically classifying commits using source code changes (e.g statement added, method removed, etc.) based on the study of Fluri's taxonomy of source code changes for object-oriented programming languages [3] into the scope of cross projects. Similarly to Levin et al., we

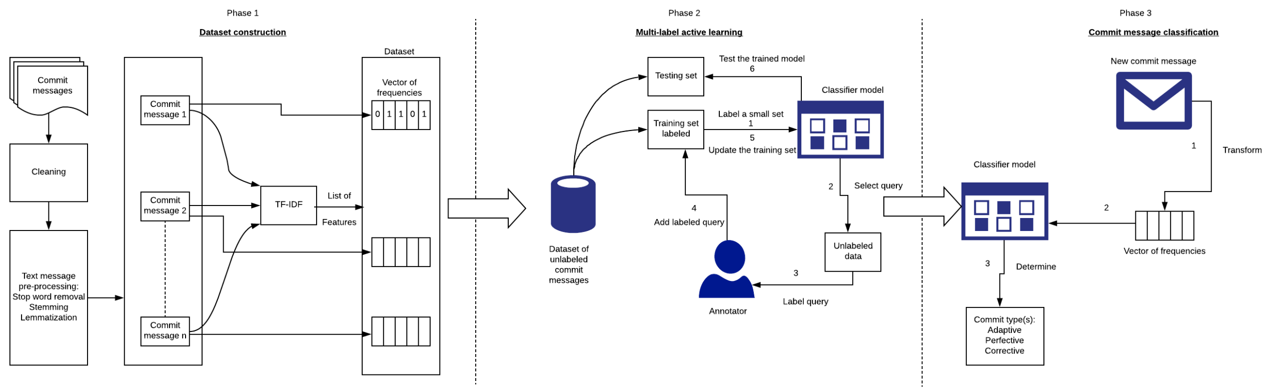

Figure 1: The commit classification framework.

as well performed commit classification into three maintenance activities using the Swanson's categories [25] and applying different technique of NLP on comments in order to get relevant information into the scope of cross projects.

4.2 Active learning for multi-label classification

Labeling a text takes a lifespan since it is expensive, subjective and error-prone, so to reduce the human effort and time in the labeling task we used the multi-label active learning approach.

The aim of Active Learning is that by repetitively increasing the size of the carefully selected labeled data, it is possible to achieve similar performance and objective via using a fully supervised data set with a simple time and cost difference time that it takes to label all the data. But, we note that in the previous studies and despite the value and the importance of this problem, we found a few research on multi-label active learning, most of them have concentrated on the aspect of single-label classification [14, 18, 24].

5 METHODOLOGY

In this section we present the overall framework of our proposed approach which is composed of three main phases : (1) Dataset construction phase which includes a text preprocessing and a feature extraction step. (2) Multi-label active learning phase during which a classifier model is built then evaluated and unlabeled instances are queried for labeling by an oracle. (3) Classification of new commit messages.

The input of Phase 1 (Dataset construction) is a set of developer's commit messages collected from GitHub for 12 different open-source projects. Hence, every commit message is considered as an instance in our final dataset. Firstly, the original commit messages undergone a preprocessing step in order to keep meaningful messages. Secondly, we used natural language preprocessing techniques to automatically extract useful keywords describing the different messages. These keywords formed the features of our dataset. The value of each feature corresponds to the frequency of it s related keywords in the original message.

The input of phase 2 (Multi-label active learning) is the dataset generated from phase 1. The output of this phase is an accurate classifier model (a logistic regression model in our case) trained on a subset of multi-labeled instances.

New commit messages will be classified in Phase 3. In fact, every new message will be transformed into a vector of numeric values, where each value corresponds to the frequency of each of one of the features that were selected in Phase 1. This vector will be then presented to the classifier model trained during Phase 2 in order to obtain it's commit type(s).

More details about the different phases are provided in what follows :

Phase1- Step1 : Text Preprocessing

Before starting this step, we cleaned up our original dataset (i.e. the set of original commit messages) by performing the following actions : (1) Removal of special characters, (2) Removal of punctuation marks, (3) Removal of short messages (with a length < 6 characters), and (4) Normalization of white-spaces.

After performing the above cleaning step, we moved to the text preprocessing step which consists of : (1) Stop-Word Removal, (2) Stemming, and (3) Lemmatization.

Phase1- Step2 : Feature Extraction

During this step, we use the set of the preprocessed commit messages as an input, and then we start extracting useful features (i.e. keywords) from text using the TF-IDF weighting scheme. This standard weighting scheme calculates the weight of the word in a text corpus by performing the product of its term frequency (TF) and inverse document frequency (IDF). A document here refers to a commit message and a corpus corresponds to the entire set of commit messages. The TF-IDF of a word t with respect to a commit message d reflects how important the word t is to a commit message d in a corpus (i.e. the entire set of commit messages).

The term frequency (TF) of a word t corresponds to the number of times t occurs in a commit message d. Hence, if we have a lot of occurrences of the same word t, we can expect an increase in its TF-IDF value. The inverse document frequency (IDF) of a word t corresponds to the logarithm (base 10) of the ratio of the total number of commit messages to the number of messages in which t

occurs. If the word t appears in many commit messages, then we can expect that it's TF-IDF will get lower.

For a given term (i.e. word) /t/ and a document (i.e. a commit message) /d/, the TF-IDF can be evaluated as follows :

$$\mathrm{TF-IDF(t,d)} = TF(t,d) \times IDF(t) \quad (1)$$

$$\mathrm{IDF(t)} = \log \frac{n_d}{DF(d,t)} \quad (2)$$

Where n_d is the total number of commit messages and DF(d,t) is the number of commit messages that contain the word t.

This approach helped us extract only useful keywords from the different commit messages. These keywords went through a refinement process where some less relevant features (e.g. domain-specific keywords, developers' information, etc.) were removed. Table 3 shows the list of the adopted features.

Each commit message m corresponds to an instance in our final dataset. Each instance was represented as a vector of numerical values where each value corresponds to the frequency of each one of the below adopted features in the message m.

Table 3: List of adopted features

Fix	Release	Master
Support	change	Use
List	Merge	Refactoring
Test	Add	Branch
Build	Set	Check
Request	Make	View
Commit	Remove	Comment
Updatesigned	Pull	Open
Cache	Revert	Translation
New	Issue	Update

Phase 2 : Multi-label active learning

Having a large dataset of unlabeled commit messages, we adopted an active learning approach with the objective of building an accurate multi-label classifier with a minimum of labeling effort, i.e., without the need to label all commit messages in the dataset which is very time-consuming.

We started with labeling a small set of instances (known as the seed) that constituted the initial training set. After choosing the multi-label approach (binary relevance with logistic regression as base classifier) as well as the multi-label query strategy to use, in each iteration of the active learning process, the query strategy queries an oracle to label a subset of the remaining unlabeled instances. New labeled instances were added to the training set. The classifier model is then trained on the updated training set and tested against a separate testing set.

For this phase, we used Libact [17], which is a python package designed for the active learning task. For the multi-label setting, Libact implements the following active learning query strategies :

— *Binary Minimization (BinMin)* : This query strategy calculates the uncertainty related to the classification of each label independently.

— *Maximal Loss Reduction with Maximal Confidence (MMC)* : This query strategy evaluates the uncertainty related to the classification of each label by calculating the difference between predictions which have a fixed formula from two different multi-label classifiers.

— *Multi-label Active Learning with Auxiliary Learner (MLALAL)* : This query strategy is considered a more general algorithm since it has a major learner that gives a binary output for each label and an auxiliary learner is used to provide information on each label to guide query decisions [17].

We choose the MLALAL query strategy since it is a generalization of both MMC and BinMin strategies. MLALAL uses a major learner (binary relevance with logistic regression as a base learner), an auxiliary learner (binary relevance with SVM as a base learner) and on the following query criteria :

— *Hamming Loss Reduction (HLR)* : This criterion assesses a strict disagreement using the decisions outputs of both classifiers to determine whether there is a disagreement or not between predicted labels.

— *Maximum Margin Reduction (MMR)* : This criterion utilizes a major classifier which outputs confidence values for the predicted classes and an auxiliary classifier that defines the contradictory predictions and outputs decisions (positive vs. negative).

— *Soft Hamming Loss Reduction (SHLR)* : This criterion tries to balance between MMR and HLR via a function that specifies the influence of each approach in the final score.

Our choice for the MLALAL query strategy with its SHLR criterion was based on a work presented in Tien et al. [17] that compared the three criteria and showed that SHLR is usually the best query criterion across different data sets and different combinations of major/auxiliary learners [17].

Phase 3 : Classification of new commit messages

Now that we have trained and built an accurate enough logistic regression model, we will use it to determine the commit class(es) for each newly posted commit message. This latter will be transformed into a vector of numeric values where each value corresponds to the term frequency of each one of the adopted features (i.e. features listed in Table 3) in the message at hand. The vector will then be handled by the classifier model and its corresponding commit type(s) are determined.

6 EXPERIMENTAL EVALUATION

In this section, we explain the dataset specification and present an experimental evaluation to challenge the effectiveness of our approach when applied to commits classification.

6.1 Data Preparation & Parameter Tuning

The evaluation of work primarily relies on its accuracy on classifying commits messages. Thus, the choice of a wide variety of potential messages is important. And so, we diversified the commit sources, i.e., projects while making sure they reflect real-world projects that contain all the possible engineering cycles including commits related to design improvement, requirements implementation, feature updates, bug fixes etc. In this context, our candidate repositories were randomly selected but conforming to the well-engineered curated GitHub Projects [20].

The selection criteria were applied using Reaper [20] and the commits were extracted using GitcProc [6], a dedicated commit crawler. Overall, we aimed to input a significantly high number of commits, we obtained 29604 commits from 12 open source projects. Details about these projects are provided in our project website [1] and in Table 4 :

Table 4: List of studied projects (project names were shortened for visibility)

Apps	#Commits	#Contributors	#Star	#Fork
geocaching	8418	93	868	491
anuto	364	24	116	36
adam.aslfms	300	38	380	90
filemanager	1344	96	2248	805
android.reddit	627	23	367	174
android.keepass	405	33	1062	334
faircode.netguard	1188	26	1768	428
fastaccess.github	939	86	3817	537
ccrama.redditslide	2526	65	991	218
mozstumbler	2159	84	540	217
keychain	4422	96	1103	366
org.videolan.vlc	6912	99	219	70

Before conducting the feature selection step, we performed stemming and lemmatization using Snowball Stemmer [2] and WordNet Lemmatizer [3]. For text vectorization parameters, we set min-document frequency to 0.01 (resp. max to 0.95) and we fixed the number of features to 50.

For machine learning, we used the scikit-learn package for Python [21]. After the text processing and feature selection, we constructed a training set of 1200 commits, manually labeled into three maintenance activities (i.e. corrective, perfective and adaptive).

Since we have to deal with a very long labeling process, we have chosen to use libact [4] python package.

6.2 Research Setting

The goal of this experiment consists of investigating the accuracy of the multi-label classification and the impact of the active learning on the rapid generation of acceptable results. Respectively, we address the following research questions :

RQ1. How effective is our approach ? The first research question explores the performance of our approach in accurately classifying commits messages. We measure the classification performance using metrics that we detail in the below-subsection.

RQ2. What is the impact of the active learning ? The purpose of this research question is to determine the effect of active learning during the classification by verifying how fast the classification converges into a satisfactory rate with a relatively low set of commits.

1. https ://smilevo.github.io
2. *https : //www.nltk.org/$_m$odules/nltk/stem/snowball.html*
3. *https : //www.nltk.org/$_m$odules/nltk/stem/wordnet.html*
4. *https : //libact.readthedocs.io/en/latest/*

6.3 Evaluation Metrics

Multi-label classification is based on finding a transformation model that maps a set of input features x to binary vectors y (assigning only the values of 0 or 1 for each element in y). There is currently no common methodology for evaluating multi-label classifications. To do the evaluation accurately, several measures from multi-class classification and information retrieval were adopted and adapted. We use the following metrics to measure performance :

Hamming Loss is the fraction of the labels that are incorrectly predicted to the total number of labels. Since it is a loss function, lower the value better is the performance [26].

$$\text{HammingLoss(h)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{L}|h(x_i)\Delta y_i| \tag{3}$$

being y_i the set of true labels associated with the example x_i, $h(x_i)$ the set of predicted labels, N the number of examples, L the total number of possible class labels and Δ the symmetric difference between the two sets.

Precision is the proportion of true positives (TP) to the sum of true positives and false positives (FP) averaged over all examples.

$$\text{Precision(h)} = \frac{TP}{TP+FP} = \frac{1}{N}\sum_{i=1}^{N}\frac{|h(x_i)\cap y_i|}{|y_i|} \tag{4}$$

Recall is the ratio of true positives to the sum of true positives and false negatives (FN) averaged over all examples.

$$\text{Recall(h)} = \frac{TP}{TP+FN} = \frac{1}{N}\sum_{i=1}^{N}\frac{|h(x_i)\cup y_i|}{|h(x_i)|} \tag{5}$$

We also use a measure that returns the harmonic mean of precision and recall, *F1-score*, defined as :

$$\text{F1} = \frac{2\times(precision\times recall)}{(precision+recall)} \tag{6}$$

where F1 score reaches its best value at 1 and worst score at 0.

6.4 Results

RQ1 : How effective is our approach ?

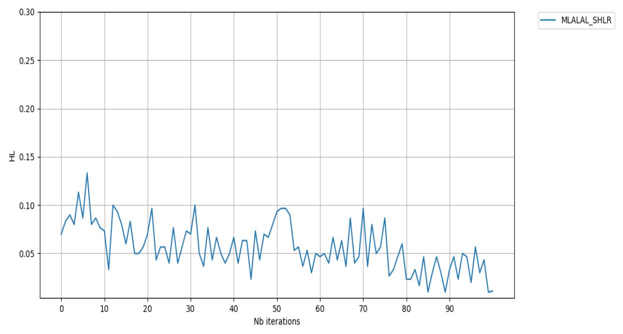

Figure 2: Evolution of the MLALAL learner Hamming Loss measure over 100 iterations.

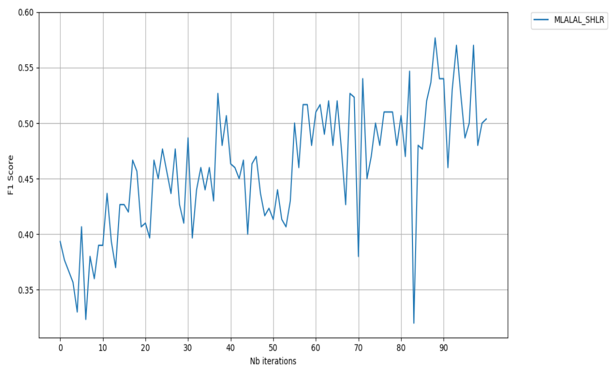

Figure 3: Evolution of the MLALAL learner accuracy in terms of F1 measure over 100 iterations.

In Figures 2 and 3, we reported the average values of the performance of the learner measured with Hamming loss and F1 loss in a train-test experiment. In our case, the HL ranged between 0.13 and 0.01 along with the 100 iterations (with a batch of 50 instances). The best HL scores are reached at the short end of the curve (iterations 80 to 100). Considering the shape of the curve it could be said that the learner actually performs well in training and gets to know better the classes.

On the other hand, we notice in Figure 3 the reverse correlation between HL and F1 scores. The increasing slope of the curve leads us to conclude that the value of the precision and recall is being balanced throughout the 100 iterations.

Furthermore, we should note that the classifier may encounter some ambiguities due to the nature of keywords' multiple meanings, which justifies the extent of variation in the curve. Nonetheless, our 45.79% average F1-score is better than the 46.32% obtained in Levin et al. [23]. This proves that our approach is quite effective when dealing with the multi-label classification. We also found 0.056 as average performance on hamming loss which is considered very promising when comparing it to the results found in Meng et al. [28].

RQ2 : What is the impact of the active learning?

The use of active learning in our approach appears to be the appropriate choice since our main goal here was to improve the multi-label classification performances.

In total, we performed 100 iterations containing 50 instances. This enabled us to manually label the subset of 5000 instances from the total number of 29604 commits. The prediction performance remains to be very satisfactory though a small number of iterations.

Also, to assess the impact of the active learning we conducted a comparison between our classifier model and 4 other scikit-learn multi-label classifiers, namely, Decision Tree (DT), K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP) and Random Forest (RF).

As shown in Figure 4, we notice that there is a huge difference in terms of results between our classifier model and the other ones that are not using active learning strategies. We also note that our classifier model has the best result in terms of HL and the second best performance in terms of F1 score. This proves the impact of active learning on the performance of the adopted approach compared to the others.

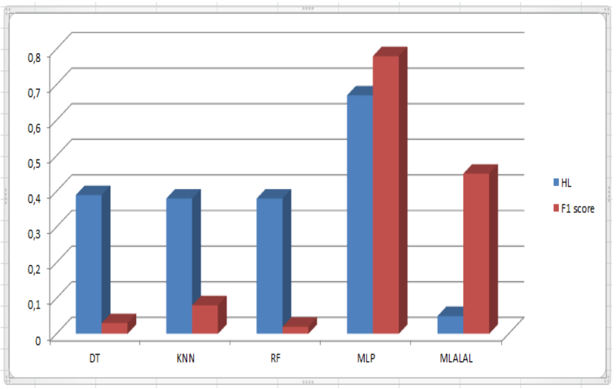

Figure 4: Comparison of performance results between classifiers.

7 DISCUSSION AND LIMITS

From the above experiments we note that MLALAL with the criterion SHLR performs promising results, especially with this diversity of data on the scope of cross-projects. Furthermore, these results in figure 4 show the importance and the efficiency of the active learning process that provides an interaction with the human to guarantee a correct classification compared to the results of the 4 other classifiers that use a fully labeled dataset instead.

However, we aware that our research may have some limitations that could have influenced the obtained results, the first is the class imbalance that was noticed while performing our study by the fact that the majority of the commits belong to the perfective class and this explains the high cost of the maintenance task.

Another interesting observation is that the perfective commits are more related to the adaptive ones and they are followed up by the corrective messages.

8 THREATS TO VALIDITY

Threats to the validity of our classification are reported in this section.

For the selection of our dataset, we randomly selected the commits from each GitHub repository of the studied projects. There is no guarantee that the selected set of commits is a good representation of all other projects, and also we cannot verify whether they represent a good distribution of commit types, and this is a threat to the internal validity of our work, and this may have a direct impact on the classification results. For that purpose, we mitigate this issue by selecting more than one project and we selected a well-distributed commits over time, performed by various developers. Also, some commits may not be significant to source code changes, such as updating documentation. so we excluded all commits which do not operate on source files and we did not consider any commits which their description was empty. The external validity of this work is linked to the generalization of our findings, and whether our findings only apply to the commits being analyzed. To mitigate this threat, we used a diverse set of well-established projects to be closer to real-world setting, and we performed our experiments across projects to challenge the performance of our classification in several scenarios.

9 CONCLUSION AND FUTURE WORK

In this paper, we propose an approach that actively learns the commit classification into maintenance activities. First of all we cleaned the set of commit messages so that we can perform the preprocessing step using the stop-word removal, stemming and lemmatization as NLP techniques, then we extracted the 50 most relevant features (i.e. keywords) from commit texts using TF-IDF and transformed into vectors of frequencies. Secondly, we used the extracted features to train our classifier model and to test it against a separate testing set after updating the training set. The output of the process of the multi-label active learning will be used into the classification of a new commit message to determine its class label after transforming this commit into a vector of frequency that is handled by the classifier model.

To assess the performance of our approach, we used 29604 as a total number of commits from 12 open source projects. The experiment results show that, we achieved the best performance in terms of hamming loss with an average of 0.05 and a good performance i.e., F1 score with an average of 45.79%.

Finally, we need to mention that our work is still subject to improvement, so for future works we plan to extend our approach and work on the changes of the nature of commits using the commit time to see the difference between them regarding their belonging to a specific category (i.e. Corrective, Perfective, Adaptive). We also aim to better classify our commits and correlate it with code changes like [23] while taking into consideration the problem of class imbalance that may impact the performance results. Moreover, to extend our approach we plan to support the automated classification of commits written in different languages relying on Babelnet for multilingual classification [4].

RÉFÉRENCES

[1] Rafi Almhana, Wiem Mkaouer, Marouane Kessentini, and Ali Ouni. 2016. Recommending relevant classes for bug reports using multi-objective search. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 286–295.

[2] Lingfeng Bao, David Lo, Xin Xia, Xinyu Wang, and Cong Tian. 2016. How android app developers manage power consumption ? : An empirical study by mining power management commits. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 37–48.

[3] B.Fluri and H.C.Gall. 2006. Classifying Change Types for Qualifying Change Couplings.. In *ICPC*. IEEE Computer Society, 35–45.

[4] Erik Boiy and Marie-Francine Moens. 2009. A machine learning approach to sentiment analysis in multilingual Web texts. *Inf. Retr.* 12, 5 (2009), 526–558.

[5] B.Settles. 2010. *Active learning literature survey.* Technical Report. University of Wisconsin.

[6] Casey Casalnuovo, Yagnik Suchak, Baishakhi Ray, and Cindy Rubio-González. 2017. GitcProc : a tool for processing and classifying GitHub commits. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 396–399.

[7] Everton Alvares Cherman, Yannis Papanikolaou, Grigorios Tsoumakas, and Maria Carolina Monard. 2017. Multi-label active learning : key issues and a novel query strategy. *Evolving Systems* (2017), 1–16.

[8] Natalia Dragan, Michael L Collard, Maen Hammad, and Jonathan I Maletic. 2011. Using stereotypes to help characterize commits. (2011).

[9] B.Yang et al. 2009. Effective multi-label active learning for text classification. In *KDD '09 : Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, New York, NY, USA, 917–926.

[10] J.Sliwerski et al. 2005. When do changes induce fixes ?. In *MSR*. ACM.

[11] J.J Amor et al. 2006. Discriminating Development Activities in Versioning Systems : A Case Study,in Proceedings PROMISE. Citeseer, 2006.

[12] Mockus et al. 2000. Identifying Reasons for Software Changes using Historic Databases.. In *ICSM*. IEEE Computer Society, 120–130.

[13] M.Fischer et al. 2003. Populating a Release History Database from Version Control and Bug Tracking Systems. *Proceedings of the International Conference on Software Maintenance* (2003).

[14] T.Luo et al. 2005. Active Learning to Recognize Multiple Types of Plankton. *Journal of Machine Learning Research* (2005), 589–613.

[15] X.Li et al. 2004. Multi-label SVM active learning for image classification.. In *ICIP*. IEEE, 2207–2210.

[16] Abram Hindle, Daniel M German, Michael W Godfrey, and Richard C Holt. 2009. Automatic classication of large changes into maintenance categories. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 30–39.

[17] C-W Hung and H-T Lin. 2011. Multi-label active learning with auxiliary learner. In *Asian conference on machine learning*. 315–332.

[18] David D. Lewis and William A. Gale. 1994. A Sequential Algorithm for Training Text Classifiers. *CoRR* (1994).

[19] Daoyuan Li, Li Li, Dongsun Kim, Tegawendé F Bissyandé, David Lo, and Yves Le Traon. 2016. Watch out for this commit ! a study of influential software changes. *arXiv preprint arXiv :1606.03266* (2016).

[20] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empirical Software Engineering* 22, 6 (2017), 3219–3253.

[21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn : Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[22] S.Levin and A.Yehudai. 2016. Using Temporal and Semantic Developer-Level Information to Predict Maintenance Activity Profiles. *CoRR* (2016).

[23] S.Levin and A.Yehudai. 2017. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes.. In *PROMISE*, Burak Turhan, David Bowes, and Emad Shihab (Eds.). ACM, 97–106.

[24] S.Tong and D.Koller. 2002. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.* 2 (2002), 45–66.

[25] E. B. Swanson. 1976. The dimensions of maintenance. IEEE Press, 492–497.

[26] Grigorios Tsoumakas and Ioannis Katakis. 2007. Multi-Label Classification : An Overview. *IJDWM* 3, 3 (2007), 1–13.

[27] T.S.Shikler and P.Robinson. 2009. Classification of complex co-occurring affective states from their expressions in speech.

[28] Rajasekar Venkatesan and Meng Joo Er. 2016. Multi-Label Classification Method Based on Extreme Learning Machines. *CoRR* abs/1608.08435 (2016).