BugHub: A Large Scale Issue Report Dataset

Renato Andrade*, Nuno Laranjeiro*, Marco Vieira†*
*University of Coimbra, CISUC, DEI, Portugal
†University of North Carolina at Charlotte, NC, USA
renatoandrade@dei.uc.pt, cnl@dei.uc.pt, marco.vieira@uncc.edu

***Abstract*—Data on issue reports have been extensively used in the literature for diverse applications. For example, in the last few years, a series of Machine Learning (ML) approaches and models have been proposed to automate software defects management processes, e.g. classification, prioritization and triage of bug fixing and implementation requests. Such works depend entirely on issue reports data and show a growing need for high-quality and heterogeneous datasets, which are not readily available in the field. This paper presents a dataset containing over 2.4 million issue reports collected from 93 projects of several natures, hosted by three tracking systems and written in 16 widely used programming languages. To demonstrate the potential of the dataset, three case studies are discussed, where more than 660,000 labelled samples are used to investigate critical aspects related to the automatic classification of issue reports using ML. Results show that our dataset has great potential and meets the quality requirements for studies that rely on issue reports data.**

***Index Terms*—datasets, issue reports, machine learning**

I. Introduction

Issue reports, frequently called *bug reports*, are semi-structured documents created by software developers, testers or users, to report situations faced when using or testing a software system [1]. A reason for creating issue reports is often the reporting of a software defect, although other cases are also frequent, such as requests for implementing new functionalities in the system, improvement requests, documentation updates and general tasks that need to be performed during the software development. Typically, issue reports are created and managed using an issue tracking system like Jira, BugZilla and GitHub. As semi-structured documents, they include both structured and plain attributes written in natural language, although some tracking systems (e.g., GitHub) only support plain text fields, such as titles and descriptions.

As procedures related to software defects management are time- and resource-consuming, automating them is a key goal of any software engineering team, but developing models to automate such processes is challenging and depends on high-quality data. Furthermore, it requires a significant amount of issue reports for training and evaluating models, especially when considering approaches based on modern word-embedding strategies and deep neural networks. The problem is that datasets of issue reports are hardly available, with the few existing options being limited to a small number of samples (e.g., [2]–[4]), collected from projects written in a single programming language and supported by a specific issue tracking system. However, a good quality dataset should consider basic aspects like its size/scale and the heterogeneity of the data in terms of programming languages, nature, complexity and size of the projects, among other relevant aspects. Creating a dataset that fulfils all these aspects is not easy.

Some issue report datasets can be found in the literature. For instance, a dataset composed of 1,158 bugs and patches extracted from 8 open-source projects was published in [2]. Also, a defect track dataset built with issue reports from both Mozilla and Eclipse projects, and focused on the bug-triage life cycle, was proposed in [5], containing 214,908 samples. Another dataset, containing 859,799 samples, created with data extracted from Chromium, Mozilla Core and Mozilla Firefox, is available in [6]. In addition, a dataset containing about 70,000 issue reports derived from 55 projects of the Apache Software Foundation is shared in [3]. Finally, a dataset containing 10,459 samples mined from GitHub repositories is available in [4]. However, many limitations can be pointed to these datasets, such as their relatively small size and lack of representativeness, which is often the result of using only a few projects, normally written in the same programming language and supported by a single issue tracker. The current lack of reasonable datasets constitutes a common problem for machine learning practitioners in this field.

In this paper **we present an issue report dataset**, available at [7], that includes about 2.4 million issue reports referring to 93 projects of different dimensions and various natures, implemented in 16 widely-used programming languages and supported by 3 issue tracking systems. The issue reports in the dataset refer to bug descriptions, improvement requests, documentation updates, etc., have different severity levels, and include a reference to the developer responsible for handling each report. The dataset was created by means of an **open-source and extensible tool** developed for this specific purpose (available at [7]).

To illustrate the potential of our dataset, **we discuss three case studies related to the automatic classification of issue**

This work has been partially supported by the FCT – Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit – UIDB/00326/2020 or project code UIDP/00326/2020; and by Project "NEXUS Pacto de Inovação – Transição Verde e Digital para Transportes, Logística e Mobilidade". ref. No. 7113, supported by the Recovery and Resilience Plan (PRR) and by the European Funds Next Generation EU, following Notice No. 02/C05-i01/2022.PC645112083-00000059 (project 53), Component 5 - Capitalization and Business Innovation - Mobilizing Agendas for Business Innovation; and by the Portuguese Foundation for Science and Technology (FCT), through the PhD grant UI/BD/151217/2021, within the scope of the project CISUC - UID/CEC/00326/2020 and by the European Social Fund, through the Regional Operational Program Centro 2020; and by INCD funded by FCT and FEDER under project 01/SAICT/2016 nº 022153.

reports using Machine Learning (ML). We study (i) the best suitable source of information (i.e, titles or descriptions) for training the models, (ii) analyse the performance of different algorithms, and (iii) investigate if the models are capable of classifying samples from projects not used in the train process. Results show that the dataset can be indeed, a helpful base for conducting different types of experiments in the context of automatic classification of issue reports. From our case studies, we discovered that, using either the report title or the description does not affect the classification outcome. Also, Support Vector Machine, Logistic Regression and Random Forest seem to be able to effectively classify issue reports. Finally, models generated based on bug reports from heterogeneous projects can, to some extent, be used to classify reports from projects not present in the training data.

The remainder of this paper is structured as follows. Section II presents the dataset and its main characteristics. Section III discusses three case studies with the goal of showing the potential of the dataset. Section IV discusses limitations and Section V concludes the paper.

II. The BugHub dataset

In this section, we present an overview of the dataset, by explaining the criteria adopted to select the issue reports and describing the data collected.

A. Criteria

Well-defined and reasonable criteria are the key for creating high-quality datasets. As we intended to collect representative, heterogeneous and useful issue reports for diverse purposes, a series of critical dimensions were taken into consideration, as shown in Table I.

Respecting the aforementioned criteria, we considered **projects** that are well-established and at the same time large enough to contain a reasonable number of issue reports, with sizes varying from thousands of lines of code (LoC) to several million, including software systems such as Firefox, TensorFlow, MariaDB and Elasticsearch, which are broadly used nowadays. We consider the selected projects to be representative due to their intrinsic different natures, complexity and sizes. For instance, Firefox is a WEB browser, while TensorFlow is a ML platform. In contrast, MariaDB is a database engine and Elasticsearch is an open search and analytics solution. Among the projects we gathered data from, one of the smallest is Apache CXF, with about 6.800 lines of code, while some of the largest ones are listed in Table II. The full list of projects included in the dataset can be found in [7].

TABLE I: Criteria considered to create the BugHub dataset.

Topic	Requirement
Projects	Different sizes and natures
Programming language	Widely-used technologies
Issue tracking system	Commonly-adopted tools
Issue reports type	*Bug-* and *non-bug-related*
Severity	Different levels
Completeness	As many attributes as possible
Filters	As necessary to ensure correctness
Minimum number of issues	As necessary to provide robustness

As **programming languages**, we considered common technologies, including Go, Java, JavaScript / TypeScript, PHP, Rust, Nix, C/C++, Python, C#, DM, Dart, Perl and Scala. These are key technologies nowadays. For example, applications such as the Minecraft game and IntelliJ IDE are written in Java, while Facebook and Slack use PHP. In addition, a great number of software systems and frameworks are created based on Python, such as Django and Salt.

The projects included in the dataset are supported by three **issue tracking systems**: Jira, BugZilla and GitHub. Jira [8] is an issue tracking software intended to support bug tracking and agile project management. It is adopted by big organizations and software communities, such as Apache Software Foundation and Reddit. BugZilla [9] is a web-based issue tracker widely used by organizations like Mozilla and Red Hat. Finally, GitHub [10] is a platform for software development and version control that includes an issue tracking system. It is used by a wide number of software development communities and companies such as Microsoft and Red Hat.

Regarding the **types** of reports, part of the dataset is labelled as *bug-* and *non-bug-related* issues (further details below), although fine-grained types can be found for several projects, such as the ones managed in GitHub, where different labels are used to categorise the issue reports. For example, Kubernetes, a well-known GitHub project, uses labels to categorise their issue reports as *bug*, *feature*, *documentation* and *support*. In terms of **severity**, several levels can be observed, such as *blocker*, *critical*, *major*, *normal*, *minor*, *trivial* and a few more.

TABLE II: Some of the biggest projects in the BugHub dataset.

Project	Language	Issues	Bugs	Non-bugs	LoC
Firefox	C/C++	25423	18607	6816	25,300,000
Mozilla Core	C/C++	164708	128608	36100	20,300,000
NextCloud Server	PHP	15392	10821	4571	9,110,000
Roslyn	C#	10248	8290	1958	5,900,000
MariaDB Server	C/C++	11746	9855	1891	4,280,000
Kibana	TypeScript	13680	11461	2219	3,230,000
Tensorflow	C/C++	6546	4912	1634	3,090,000
QGis	C/C++	24080	20543	3537	2,190,000
Godot	C/C++	23727	21105	2622	1,590,000
MongoDB Server	C/C++	28641	13730	14911	1,590,000
Spring Framework	Java	12734	4440	8294	1,420,000
Elasticsearch	Java	20026	9605	10421	1,200,000
Bazel	Java	3283	2110	1173	1,110,000
Mozilla NSS	C/C++	6493	4144	2349	1,080,000
Symfony	PHP	16759	11602	5157	1,030,000
SeaMonkey	C/C++	9946	8765	1181	1,020,000

To ensure **completeness**, we collected all available attributes supported by each issue tracking system, as those can be important for future research using the BugHub dataset. To improve the trustworthiness of the samples classified as bugs and non-bugs, we used **filters** to consider only reports that match *FIXED* and *RESOLVED* cases, thus ensuring that they were previously adequately analysed by the developers.

To make the dataset representative and heterogeneous, we considered only projects that have a **minimum number of issues** available in the corresponding issue tracking system. We name this parameter as n and analysed different values for it. Considering that the *bug* versus *non-bug* categorisation is typically the basis for the automatic classification of issue reports, we investigated the literature to understand the dimensionality used for such purpose, which refers to the number of features or attributes obtained from plain text to create ML-based models. As studies in this context typically adopt between 100 and 150 dimensions (e.g., [11], [12]), we decided that 1,000 issues per class would be a reasonable amount of reports to consider (a number 6 to 10 times greater than the usually adopted dimensionality).

B. Dataset details

To create the BugHub dataset, we opted for using Jira, BugZilla and GitHub as source of information, as they are widely used by well-known projects and provide public REST APIs that are powerful and well-documented. The acquisition process, run between April 2021 and May 2023, resulted in 2,462,666 unique issue reports from 93 projects (i.e., 34 from Jira, 8 from BugZilla and 51 from GitHub) being processed. As mentioned before, such projects are of different categories, including (but not limited to) operating systems, programming languages, issue tracking systems, mail clients, HTTP servers, web browsers, etc.

From the 2,462,666 samples in the dataset, 1,419,886 are labelled as *bug* or *non-bug*. These are particularly useful for automatic classification processes. The dataset also contains 941,905 samples divided into several different severity levels, including *blocker*, *critical*, *major*, *normal*, *minor* and *trivial*. These samples are particularly a handful for creating ML models to perform automatic prioritization of issue reports. To support triage, labelled samples are assigned to the developers who worked on the issue report solution. The unlabelled samples in the dataset (1,042,780) are intended to train unsupervised ML algorithms and word-embedding-based models and are also helpful for anomaly detection.

As shown in Figure 1, not all samples are categorized as *bugs* and *non-bugs*, and some are not classified according to their criticality. This is to ensure that classified samples are correctly labelled. In fact, most samples in the dataset labelled as bugs/non-bugs refer to fixed and resolved issue reports, given that these are the ones that have been reviewed by developers (ensuring that they are correctly classified). Furthermore, samples from GitHub are classified using tags that can be defined by each project team. This means that each project may use a different taxonomy for identifying bugs and non-bugs and for defining criticality. In some cases, it is difficult to be sure about what each tag represents, which creates uncertainties in labeling the samples. We decided not to label such samples but still include them in the dataset.

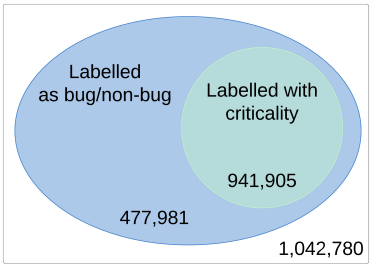

Fig. 1: Number of issue reports in each category.

In terms of programming language, the BugHub dataset includes widely used technologies such as Java, Python, Go, PHP and C/C++, with this last one being a combination of C and C++ (as these share many similarities). We also considered JavaScript and TypeScript as one single language (JS/TS), given that most projects based on these technologies usually adopt both of them. Figure 2 shows the distribution of issue reports in the dataset by programming language. The x axis represents the $\log_2 numberofissues$, while the number in each bar represents the total amount of samples in the corresponding programming language. The Figure indicates that C/C++ projects are the ones contributing with more reports to the dataset, followed by JS/TS, Go, Java, Python and Rust.

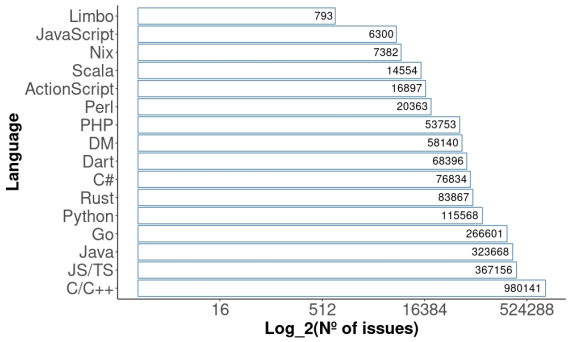

Fig. 2: Number issue reports per programming language.

Figure 3 shows the overall amount of *bug*- and *non-bug-related* reports per programming language (considering only the samples labeled as *bug*/*non-bug*). As shown, projects tend to have fewer issues not related to software defects in comparison to the ones related to bugs. In general, C/C++ and Java are the programming technologies that appear associated with a higher number of samples in this part of the dataset.

Concerning the severity of the issue reports, Figure 4 shows the amount of samples in each of the main levels, which refer to projects extracted from both Jira and BugZilla (as mentioned before, GitHub uses a different method to define priorities). As shown, *normal* severity is the one where most

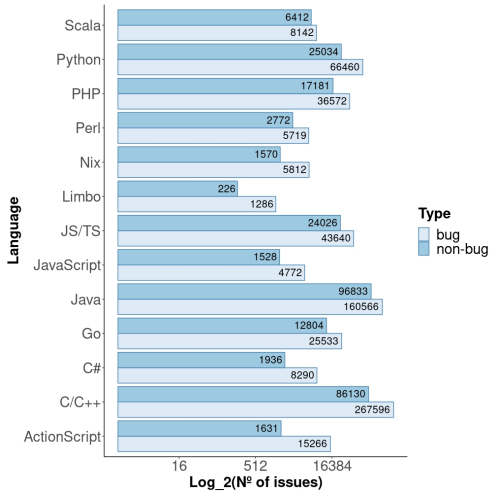

Fig. 3: Total of issue reports per programming language.

issue reports lie, as this is the default level found in most projects. However, a relatively high number of issue reports is categorised as *critical* and *major*, while *blocker* is the level associated with fewer samples.

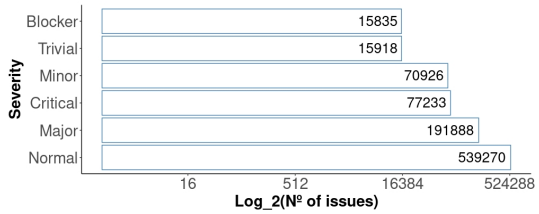

Fig. 4: Volume of issue reports per severity level.

The dataset is publicly available as a relational database dump (i.e., a *.sql* compressed file) [7]. It contains a table with issue reports from BugZilla labelled with severity levels and another table with unlabeled samples from GitHub. More importantly, there is a table with the reports collected from Jira, BugZilla and GitHub that are categorized as *bugs* and *non-bugs*. In this table, issue reports from Jira are also labelled with priorities. Finally, the dump contains a series of functions to export all data categorized as *bug-* and *non-bug-related*, according to each particular issue tracking system. The views and functions are included in the dump file to allow for simple and effective ways of exploring and exporting the data.

III. Case Studies: Automatic Classification of Issue Reports

In this section, we present three case studies on the automatic classification of issue reports using ML, with the following objectives:

- **Case study 1**: Investigate if the dataset can be used to analyse which content from the issue reports (i.e., titles *vs.* detailed description) is best suitable to create models for automatic classification of issue reports.
- **Case study 2**: Understand if the dataset would support experiments to reach solid conclusions when studding the performance of different classifiers in this context.
- **Case study 3**: Investigate how well ML performs on the automatic classification of issue reports when handling data not involved in the training process.

A. Motivation and Methodology

Software defects (the so-called *bugs*) constitute a significant threat to the safety and reliability of software systems. Although bugs are reported in a large scale daily in many software projects, not all issues registered are related to software defects. In fact, the literature shows that more than 30% of all issue reports submitted as bugs to open-source projects are actually non-bug-related, instead of requests associated with software defects [1].

In the last few years, ML-based approaches have been proposed to automatically classify issue reports into bug- and non-bug-related, or other fine-grained categories. The proposed methods (e.g., [13] and [14]) are mainly based on supervised learning, which means that they depend on labelled issue reports for training the algorithms. However, achieving reasonable performance is challenging when handling the automatic classification of issue reports, mostly due to difficulties in acquiring reliable data to build models.

Our case studies focus on critical aspects related to the automatic classification of issue reports using ML, with the primary objective of assessing the usefulness and quality of the BugHub dataset. The following design decisions drive the case studies:

- **Data:** We use a subset of the BugHub dataset containing over 660,000 issue reports labelled as *bug-* and *non-bug-related* (detailed in [15]). This subset includes data extracted from 52 projects developed in 10 different programming languages, supported by Jira, BugZilla and GitHub tracking systems.
- **Text preprocessing:** We used techniques widely-used in Natural Language Processing (NLP) problems, namely *tokenization*, *stop-words removal* and *lemmatization* [16].
- **Feature generation:** We opted for a bag-of-word (BoW) approach, weighted by term-frequency with inverse document frequency (TF-IDF) [17] as this approach is widely used in this context.
- **Dimensionality reduction:** We adopted chi-squared method , which is a ranking technique [18] that require less processing power than others.
- **Performance assessment:** We decided to use *F-measure* [19], which is the harmonic mean between *precision* and *recall*, as it is a suitable metric for analysing results from imbalanced testing.

We adopted the following general practices. We randomly select 70% of the data for training and the remaining 30% for testing the models. For training, we use balanced data, while imbalanced samples are used for testing (i.e., the original proportion among classes as it appears in the dataset). We execute 30 train/test repetitions for each experiment to study

how the results may vary. To analyse the results, we resort to visual observation and statistical testing. To check the assumptions for applying parametric tests, we use two different methods: *i*) a Shapiro–Wilk test, to check whether the results follow a normal distribution or not; and *ii*) Levene's and Fligner-Killeen test to verify if the variance is similar among the samples being tested. As our results do not fulfil the requirements to carry out parametric tests, we use a pairwise version of the *Wilcoxon rank sum test* with *p-values* adjusted by the *Bonferroni* method with a typical value of $\alpha = 0.05$, for formal hypothesis testing [20].

B. Case Study 1: Titles vs. Descriptions

In this case study we investigate which content from the issue reports is best suitable to train the models. We analyse whether using the full textual description produces better results than using titles. To avoid the *curse of dimensionality*, a common problem when using BoW approaches, we also analyse what would be the most appropriate number of features (i.e., dimensions) to use. For our experiments, we opted for using the Logistic Regression (LR), as hyperparameter optimization is not a critical issue for such a classifier. We also resorted to the 52 projects contained in the subset of issue reports mentioned previously.

To investigate the ideal number of dimensions, we trained LR models with both titles and descriptions, adopting the following number of dimensions: {*50, 100, 150, 200, 250, ..., 500*}. Figure 5 shows the resulting mean F-measures. By looking at the chart, it can be seen that the performance of the models varies more expressively between 50 and 250 dimensions. When the number of dimensions grows to more than 250, differences in the F-measure seem to be smaller. Also noticeable in the figure are the error bars, which are relatively large as the models reach different F-measure values depending on the project (e.g., for 250 dimensions, the F-measure of models from Spring-boot project is about 0.83 in some cases, while models from Firefox are associated with a mean F-measure of 0.57).

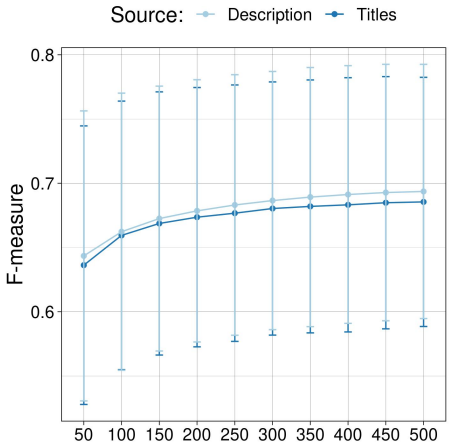

Fig. 5: Mean F-measure by number of dimensions.

After a visual analysis, we compared each dimension with the others above through a formal hypothesis testing procedure, to find out if the F-measure values vary significantly with the increasing number of dimensions. As a result, we found out that **there are no statistically significant differences in the performance of models built based on 250 dimensions and those created by using a higher number of dimensions, for both titles and descriptions**.

A further investigation was conducted to analyse the reasons behind our findings. The importance of each term in the BoW reveals that a small amount of words is frequently connected to a higher relevance in the feature selection process, before the model creation. We noticed that the average weight of the first 250 selected words is about 30% higher than the scores of the words in the range going from 250 to 500 dimensions. We can highlight one example extracted randomly from our experiments carried out with project AngularJS, in which we rank a set of terms extracted from titles by their TF-IDF weights . We then observed that the bi-grams *properly handle* and *implement detach* lie in two of the first positions and are associated with scores of 7.8 and 3.3, respectively. In contrast, the term *group* appears in position 255, with a weight of only 0.2. In our analysis, models built based on 250 dimensions leverage the most important terms. Thus, adding more features is not helpful for a significant performance improvement. Instead, many models using more than 250 dimensions suffer performance losses as they become too adapted to their training sets.

We then proceeded to a different test, in which the objective was to verify whether using titles leads to better performance than resorting to descriptions or vice-versa. We used the F-measures obtained with 250-dimension models, according to the results of the previous hypothesis testing procedure. We statistically compared both contents and as a result, we observed that **F-measure values are not significantly different whether using titles or descriptions**. Figure 6 shows the data tested. We can observe that, although the descriptions are associated with a higher mean F-measure, the interquartile F-measure values for titles lie in a similar position.

When analysing the reasons behind such a result, we

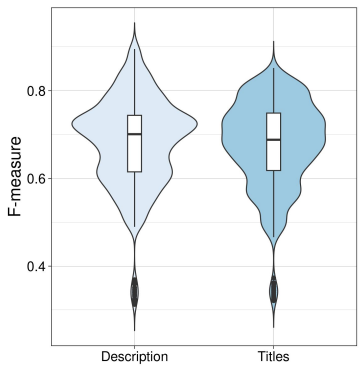

Fig. 6: Performance comparison - titles *vs.* descriptions.

observed that a few terms present in titles are connected to elevated importance in the decision process. In contrast, most words in descriptions have no relevance to the models. As descriptions are usually longer than titles, they require more processing power and might not be the best suitable choice to create models for the automatic classification of issue reports. When focusing on the weights given by TF-IDF method to each word, it becomes clear that terms from titles are about 20% higher than terms in descriptions. In practice, performance losses that would occur due to the small number of words present in titles are compensated by the strong weights their terms tend to have, leading models built based on titles to a performance that is comparable to that achieved by models created by using descriptions.

This case study allows us to assess the quality of the BugHub dataset in different ways. First, the heterogeneity of the data is imperative to achieve effective results, as using projects of different natures, sizes and complexities reinforces the reliability and representativity of the experiments. Secondly, the reasonable number of issue reports used is imperative to reach solid conclusions, as using a small amount of samples may not be enough to represent the variety of real scenarios we come across in this context. Lastly, using issue reports that have been previously analysed by developers to guarantee their correctness is the basis for conducting not only this type of study but also future research in this field.

C. Case Study 2: Performance of Classifiers

Our objective in this case study is to verify if the samples from the BugHub dataset meet the requirements to investigate which algorithm is capable of producing models that present an overall higher performance when facing the automatic classification of issue reports. We selected five well-known algorithms: Random Forest (RF), an ensemble classifier; Naıve Bayes (NB), which is a Bayesian classifier; Support Vector Machine (SVM), a maximal-margin classifier; k-Nearest Neighbor classifier (k-NN), known to be a lazy learner; and Logistic Regression (LR), a probabilistic approach [21]. To conduct the experiments for this research question, we resorted to the full subset mentioned in Subsection III-A, which includes 52 projects as heterogeneous as possible in terms of nature.

To carry out our experiments, for each of the five algorithms, we executed 30 train/test sequences per project. Thus, we ended up with 7,800 resulting F-measures to be analysed (i.e., 1,560 outcomes per algorithm). Figure 7 presents a visual comparison of the results. To verify whether the differences in the F-measures were significant or not, we applied the usual statistical procedures comparing each algorithm with the remaining four. Based on our results, we concluded that **the performance reached by SVM, RF and LR classifiers is not significantly different from each other, while NB and KNN models tend to perform poorly in comparison to the remaining algorithms.**

Although we are presenting a summarized version of our results, we believe that it is enough to demonstrate the utility of the dataset. The aforementioned case study allowed us to conclude that issue reports from the BugHub dataset are heterogeneous and representative enough to reach solid conclusions when assessing the performance of models built to perform automatic classification of issue reports. In general, this type of study requires data that can reflect as much as possible real case scenarios, which implies using samples from projects that are intrinsically different in many aspects, such as programming languages, sizes, objectives, issue tracking systems used to support them and so on.

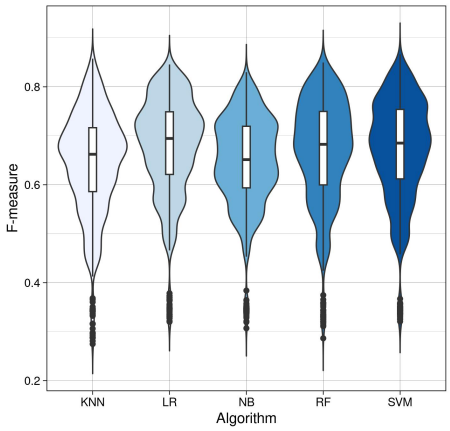

Fig. 7: F-score comparison between algorithms.

D. Case Study 3: Cross-project Classification

The main objective of this case study is to assess if projects in the BugHub dataset can be used for experiments targeting the investigation of how well ML algorithms can carry out the automatic classification of issue reports when handling data not involved in the training. To achieve such a purpose, we adopted five projects written in a common programming language (Java) and supported by the same tracking system (GitHub). We do so to prevent possible fluctuations in the performance due to the use of different programming languages and issue trackers. For this case study, we used the following projects: Bazel, Elasticsearch, Netty, Spring Boot and Spring Framework. We used Support Vector Machine (SVM) as a classifier to provide us with the needed results.

To create a baseline, we started by building models trained and tested with issue reports coming from the same project. To avoid bias towards large software systems, we extracted the same amount of issue reports from each project. Concretely, the following procedure was followed for each system: first, 2,000 reports were randomly selected and divided into the previously defined proportion for training/testing (i.e., 1,400 samples for training and the remaining for testing). We also ensure that classes are balanced in the training set (i.e., 50% bugs and 50% non-bugs).

Figure 8 shows the F-measure values obtained when training and testing with data from the same project. As demonstrated, the performance of the models seems to differ from project to

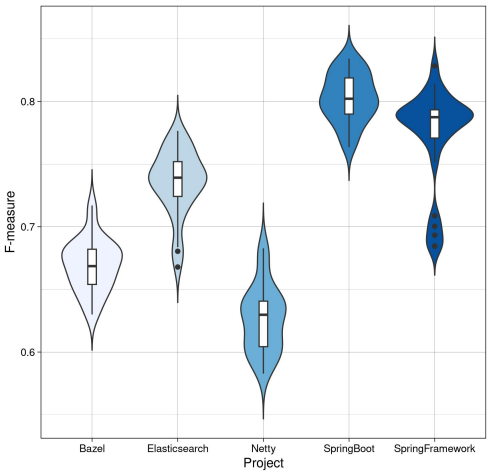

Fig. 8: Results for training and testing with the same project.

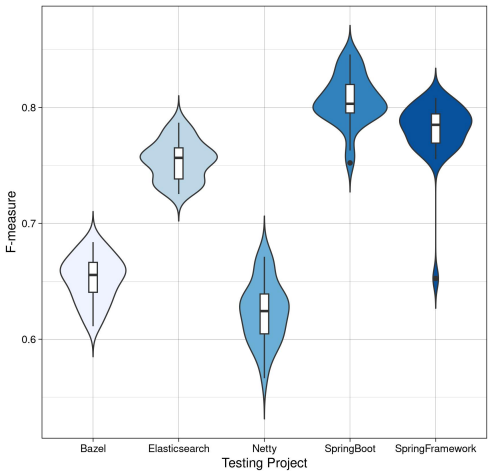

Fig. 9: Cross-project classification with SVM.

project. Spring Boot is the system from which the algorithms reach higher scores, and Netty is the software associated with lower F-measures. We resort to statistical procedures to verify whether the differences in the F-measure values are significant or not. The results indicate that they are significantly different between the projects. In summary, we can state that *even when using projects written in the same programming language and supported by the same issue tracking system, performance varies significantly*.

In the next step, we trained models using data from four projects and tested them with the remaining one. We refer to this procedure as *cross-project classification*. Considering that we were using five projects and we ran 30 train/test sequences per testing project, we ended up with 150 sets of results. This number is not different from what we had in the experiments previously conducted in the first part of the case study. We start the procedure by randomly selecting 6,000 issue reports from the four projects (i.e., 1,500 samples per project, balanced into 50% bug- and 50% non-bug-related). From the fifth project (used as a testing project), a random selection of 600 samples (keeping the original proportion of each class) is then carried out. By doing so, we follow a round-robin manner, passing each of the five projects to the testing phase.

Figure 9 shows the F-measure resulting from SVM for each testing project. When comparing it with Figure 8, we can see the same pattern, although some differences can be noticed in the shape that represents the F-measures distribution of the projects. We resort to the usual statistical methods to test whether the F-measure values are significantly different according to the testing project or not. Results support the claim that *when performing cross-project classification, the performance of the models varies significantly depending on the testing project*.

The final step in the case study was to compare results from cross-project classification with those obtained when training and testing with issue reports from the same project (i.e., the first part of the case study). This is basically a comparison of pairs of projects shown in Figures 9 and 8, as differences can be complicated to distinguish visually. The usual statistical methods indicated significant differences in two cases: Elasticsearch and Bazel projects. Yet, for the remaining three projects, differences in the F-measures were not statistically significant. Based on our results, we can conclude that *the performance reached by the models when performing cross-project classification may not always be significantly different from the performance of models trained and tested with issue reports from the same project*.

To explain our results, we analysed the terms kept during the feature selection process in each project. Then we compared the selected terms among all projects. Our analysis revealed that the amount of selected words which are shared between any two projects written in the same programming language and supported by the same issue tracking system is on average, 9%. This means that only a few terms are simultaneously relevant for two random projects. Our findings support the idea that, in theory, increasing the number of projects in the training set might result in higher performance, as long as the same programming language and the same issue tracking system are respected.

This case study was conducted by using five different projects from the BugHub dataset holding particular characteristics (i.e., programming language and tracking system). Besides the importance of such similarities, this type of investigation also requires projects which are different in some aspects, such natures and complexities. The dataset showed itself sufficiently heterogeneous to allow us to choose projects meeting the requirements for our experiments. This was possible due to the large amount of samples included in the dataset with similarities and differences, indicating that it is capable of supporting a reasonable number of studies related not only to cross-project classification but also to the use of ML to automate processes in software defect management.

IV. Limitations and Threats to Validity

BugHub is a broad and heterogeneous dataset; however, it has some limitations, namely regarding size and variety. We discuss such limitations in the following paragraphs.

A set of criteria were established to create the dataset. Although the set is diverse, we may have missed relevant criteria, we may have included less relevant ones, or may have underachieved some of the identified ones. Indeed, understanding to what extent a criterion is achieved or not is quite difficult, as current literature is scarce on this kind of information. The important properties to reach were approached in discussion with experienced researchers, but most of all, the supporting infrastructure allows for updates.

From the 2,462,666 unique samples included on it, the amount of samples marked with *bug* or *non-bug* is 1,419,886, a number which we still intend to increase in the future. Of the 1,419,886 million issue reports, 927,737 samples have been marked with *FIXED* and *RESOLVED* by the corresponding project developers. The remaining 492,149 were not marked with *FIXED* and *RESOLVED* at the time of collection, which means they had not been (at that moment) analyzed by developers. In turn, this means that our dataset may benefit from being updated in the future so that these reports are replaced by updated versions confirmed by developers.

BugHub contains a reasonable number of issue reports marked with different severity levels (i.e., a total of 941,905). However, this particular set has reports extracted from only Jira and BugZilla. The dataset may benefit from further variety.

We developed a tool named ITSDataCollector to automate the data acquisition process (available at [7]). The tool can be used to overcome the limitations presented in this section. For instance, whenever new issue reports are added to a project, the tool can be easily used to update the database with the new data. It is prepared to collect new issue reports as it keeps control information in the database, allowing it to start from the last sample gathered from a specific project.

Regarding threats to the validity of our case studies, to avoid the *curse of dimensionality*, we adopted the chi-squared method for feature selection. For *hyperparameter optimization*, we ran the optimization several times using random samples from different projects. To ensure *randomization*, in each train/test sequence, we chose a number of samples lower than the total of examples available. To avoid *bias and noise*, we carried out each train/test 30 times. Regarding *data quality*, we used preprocessing techniques that were already tested in previous studies. To guarantee data *representativity*, we acquire at least 1,000 examples per class in each project we used. Finally, we highlight that our case studies resulted in much more information that is not presented here due to the limited space.

V. Conclusion

This paper described a dataset comprising about 2,4 million issue reports, gathered from various tracking systems and including different projects written in several programming languages. We showed how the dataset can be useful for research in issue classification by using it in three different scenarios. The dataset is now publicly available at [7] for open use in future research in the classification of issue reports, in various dimensions (e.g., prioritization, triage).

In future work, we will further extend the dataset and address limitations related to unreviewed reports. We also intend to use it for creating machine learning models for severity classification of issue reports.

References

[1] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y. G. Guéhéneuc, "Is it a bug or an enhancement? A text-based approach to classify change requests," *Proceedings of the 2008 Conference of the Center for Advanced Studies, CASCON'08*, 2008.

[2] R. K. Saha, Y. Lyu, W. Lam, H. Yoshida, and M. R. Prasad, "Bugs.jar: A large-scale, diverse dataset of real-world Java bugs," *Proceedings - International Conference on Software Engineering*, pp. 10–13, 2018.

[3] R. Vieira, A. Da Silva, L. Rocha, and J. P. Gomes, "From reports to bug-fix commits: A 10 years dataset of bug-fixing activity from 55 Apache's Open source projects," *ACM International Conference Proceeding Series*, pp. 80–89, 2019.

[4] T. Hirsch and B. Hofer, "Root cause prediction based on bug reports," *Proceedings - 2020 IEEE 31st International Symposium on Software Reliability Engineering Workshops, ISSREW 2020*, pp. 171–176, 2020.

[5] A. Lamkanfi, J. Pérez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information," *IEEE International Working Conference on Mining Software Repositories*, pp. 203–206, 2013.

[6] S. Mani, A. Sankaran, and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging," *ACM International Conference Proceeding Series*, pp. 171–179, 2019.

[7] R. Andrade, N. Laranjeiro, and M. Vieira, "BugHub: A Large Scale Issue Report Dataset - Supplementary Material," Oct. 2023. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.10028952

[8] "Jira Software." [Online]. Available: https://www.atlassian.com/software/jira

[9] "Bugzilla." [Online]. Available: https://www.bugzilla.org

[10] "GitHub." [Online]. Available: https://github.com

[11] N. Pingclasai, H. Hata, and K. I. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, vol. 2, pp. 13–18, 2013.

[12] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1770

[13] A. Kukkar and R. Mohana, "A supervised bug report classification with incorporate and textual field knowledge," *Procedia Computer Science*, vol. 132, pp. 352–361, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.05.194

[14] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, "Bug or not? bug report classification using n-gram idf," *Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, pp. 534–538, 2017.

[15] R. Andrade, C. Teixeira, N. Laranjeiro, and M. Vieira, "An Empirical Study on the Classification of Bug Reports Using Machine Learning Algorithms - Supplementary Material," Nov. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7377402

[16] J. Thanaki, *Python Natural Language Processing*, 2017.

[17] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] J. Novaković, P. Strbac, and D. Bulatović, "Toward optimal feature selection using ranking methods and classification algorithms," *Yugoslav Journal of Operations Research*, vol. 21, no. 1, pp. 119–135, 2011.

[19] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

[20] D. J. Denis, *Univariate, Bivariate, and Multivariate Statistics Using R: Quantitative Tools for Data Analysis and Data Science*. Wiley, 2020.

[21] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge: Cambridge university press, 2014.