ORIGINAL RESEARCH PAPER

Automated Issue Assignment using Topic Modeling on Jira Issue Tracking Data

Themistoklis Diamantopoulos | Nikolaos Saoulidis | Andreas Symeonidis

Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, Thessaloniki, Greece

Correspondence
Themistoklis Diamantopoulos,
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, GR54124, Greece
Email: thdiaman@issel.ee.auth.gr

As more and more software teams use online issue tracking systems to collaborate on software projects, the accurate assignment of new issues to the most suitable contributors may have significant impact on the success of the project. As a result, several research efforts have been directed towards automating this process to save considerable time and effort. Most approaches, however, focus mainly on software bugs and employ models that do not sufficiently take into account the semantics and the non-textual metadata of issues and/or produce models that may require manual tuning. In this work, we design a methodology that extracts both textual and non-textual features from different types of issues, providing a Jira dataset that involves not only bugs but also new features, issues related to documentation, patches, etc. Moreover, we effectively capture the semantics of issue text by employing a topic modeling technique that is optimized using the assignment result. Finally, our methodology aggregates probabilities from a set of individual models to provide the final assignment. Upon evaluating our approach in an automated issue assignment setting using a dataset of Jira issues, we conclude that it can be effective for automated issue assignment.

KEYWORDS
Mining Software Repositories, Project Management, Task Management, Jira Issues, Topic Modeling

1 | INTRODUCTION

The introduction of the open-source paradigm and the evolution of online services has lately transformed software development into a highly collaborative process. Software teams nowadays use online facilities not only to host their code and software artefacts but also to effectively monitor the software development process. In this context, several issue tracking systems have been deployed online by large open-source organizations, such as the Eclipse Foundation, the Mozilla Foundation (that both use Bugzilla[1]), or the Apache Software Foundation (that uses Jira[2]). When having multiple contributors and multiple projects, these services provide effective ways to keep track of tasks, prioritize and assign new features, resolve bugs, and set deadlines for crafting new releases. Moreover, the (meta)data produced by these services are abundant, and can be mined to further improve the software development process.

When a new issue is created in an issue tracking system, a member of the team known as *triager* has to assign it to another team member/contributor of the project. Optimally assigning tasks to team members is far from trivial, as it requires taking into account the past experience and the current status of each team member individually, and further having clear knowledge of the task at hand and the project as a whole. Keeping track of the specifics of team members and product features is not always straightforward, especially as a project grows and undergoes changes. Thus, the process of determining the most appropriate contributor to undertake a task may require considerable time and effort.

As a result, several approaches have been developed to automate issue assignment. Most of these approaches use data from bug tracking systems and employ different classification algorithms to recommend the assignment of new issues/bugs based on past assignments. Each approach may extract textual and/or non-textual data from bug reports and represent them in different ways, including e.g. vector space models or word embeddings. After that, various machine learning [1, 2, 3, 4] or even deep learning [5, 6] algorithms are used to provide assignment recommendations. Furthermore, there are also approaches employing topic modeling techniques [7, 8, 9, 10] in order to further capture the semantics of issues and/or build developer profiles that can be helpful for making the assignment.

Several of these approaches focus on bug tracking systems, thus they are most effective in the narrow scope of bugs, neglecting issues/tasks that refer to new features or documentation. Certain methods are also typically limited to textual attributes (i.e. issue titles and descriptions) and employ models that do not sufficiently capture their underlying semantics. Topic modeling approaches, on the other hand, extract semantically rich topics, however they also may not incorporate non-textual data (i.e. issue labels, priorities, types, etc.), while they also may not optimize the parameters of the model (i.e. the number of topics).

In this work, we design a system that performs automated issue assignment while confronting the aforementioned limitations. Our system is built using a dataset extracted from the Jira infrastructure of the Apache Software Foundation, which comprises issues involving not only bugs but also development and documentation-related issues. We employ a topic modeling technique that is optimized using the assignment/triaging result, thus making sure that the extracted topics (and their top terms) are tailored to the automated assignment. Finally, apart from issue titles and issue descriptions, our system combines prediction models built for non-textual characteristics, include the type, the priority, and the labels of the issues, in order to produce more effective recommendations.

The rest of this paper is organized as follows. Section 2 reviews the related work in the area of automated issue/bug assignment. Our methodology for an automated issue assignment system is presented in Section 3. Section 4 evaluates our approach on a set of software projects, assessing the effect of different variables and the added value of topic modeling. Finally, Section 5 discusses the threats to the validity of our approach and evaluation, while Section 6 concludes this work and provides useful insight for future research.

[1] `https://www.bugzilla.org/`

[2] `https://www.atlassian.com/software/jira`

2 | RELATED WORK

As already mentioned, the evolution of software development and the introduction of online collaboration platforms has driven the research community towards optimizing the software development process. In this context, several research efforts have been directed towards employing issue tracking data for various goals, such as identifying developer characteristics and/or building profiles [11, 12], prioritizing issues/requirements [13, 14, 15, 16], or even determining the severity of bugs [17, 18, 19]. In this work, we provide a dataset that can be used for assessing several different challenges, and specifically focus on the challenge of automated issue assignment (known also as "bug/issue triaging").

The majority of the proposed methodologies in the area of automated issue assignment extract data from issue tracking systems, such as Bugzilla, and train multiclass classification algorithms in order to identify the developer that should undertake a new issue, given past assignments. One of the first approaches in this area is that of Murphy and Cubranic [1]. The authors extract bug reports from the Eclipse project, employ a vector space representation to model their textual description, and use Naïve Bayes to classify them to different developers. Anvik et al. [2] extend this approach by including more information, such as e.g. the current workload of the project contributors, and employ Support Vector Machines (SVM), managing to significantly improve the accuracy of the assignments. Their system is semi-automated and provides a list of the most suitable developers for each bug report, ranked according to their relevance, allowing the triager to make the final assignment.

Matsoukas et al. [3] further employ information about issue comments as well as commits of software projects. To do so, they use a dataset extracted from GitHub [20] and employ a neural network that aggregates features probabilities in order to automate the assignment of GitHub issues. Similarly, Alkhazi et al. [21] extract information from commit messages and treat it as domain knowledge to create developer profiling features, which are thereafter used to determine the most suitable developers for resolving a bug report. An interesting alternative is proposed by Sajedi-Badashian and Stroulia [22]. The authors extract programming keywords from bug reports, and create a text model that weights them using term frequencies from Stack Overflow. This way, the system can emphasize technical terms, which are arguably more useful for resolving bugs.

Xuan et al. [4] note that information regarding assignees is not always available in issue tracking systems, thus they use Expectation Maximization to handle missing labels and Naïve Bayes to make the assignments. Apart from the aforementioned machine learning approaches, lately there are also approaches employing deep learning techniques. The focus in this case is on text representation; as indicated by current research [23], textual features may even decrease (instead of increase) the effectiveness of bug triaging, when the employed representations lack semantics. As a result, contemporary approaches use representations based on word embeddings [24, 25], instead of the typical bag-of-words/vector space model representations, while there are also approaches that employ attention networks [26] and pretrained language models [27]. The classification is performed using Recurrent [5, 26] or Convolutional Neural Networks [6, 24]. An interesting approach in this category is the Software Bug Report Network (SBRNet) of Mohsin et al. [28], which functions as a knowledge network. The authors build a two-stage model that first classifies bugs into different categories and then assigns project contributors to classified bug types using the information inferred by the network.

Another interesting line of research, which is most relevant to this work, is that of employing topic modeling techniques to create topic-based developer profiles and/or group bug reports into topics. The information extracted from issue tracking systems can be quite rich in semantics, therefore it can be used for improving automated triaging. One of the first approaches in this area is that of Ahsan et al. [7], who applied Latent Semantic Indexing (LSI) on the titles and descriptions of bug reports. The resulting term-document matrix

was subsequently used along with different classifiers to classify bug reports to developers, with SVM achieving the best accuracy. A similar approach is followed by Yang et al. [8] who use Latent Dirichlet Allocation (LDA) and create topics corresponding to different bug reports. So, when a new bug report arrives, they first determine its topic(s) and, thus, find similar bug reports, and then utilize the features of these bug reports to perform the classification.

Naguib et al. [9] also use LDA and leverage the activities of developers throughout the project (i.e. bugs assigned, resolved or reviewed) to create personalized profiles. These profiles are then compared to newly created bug reports in order to produce a ranked list of relevant developers. An interesting approach is also proposed by Xia et al. [10], who employ a specialized topic modeling technique, named Multi-feature Topic Model (MTM), that extends LDA by taking into account the components and the products of each bug report. The topic distribution of each new bug report is subsequently used to find the developer who is most relevant to the bug's topics, and therefore most appropriate to resolve the bug itself.

Another interesting challenge in the broader area of bug triaging is that of "bug tossing". Bug tossing happens in cases where a bug has to be re-assigned to a different developer as it was not be resolved in the first place. The approaches in this area focus on the tossing sequences of bug reports and employ techniques such as Markov chains or sequence-to-sequence models to recommend the most suitable developer and/or reduce the length of tossing paths (i.e. number of tossings happening within a software project) [29, 30, 31]. A similar direction is that of *scheduling* bugs, i.e. evaluating all possible orders with which bugs should be resolved [32]. The main rationale in this case lies in taking into account not only the profiles but also the workload of the developers. Though interesting, these approaches deviate from the scope of this work.

Finally, we note that certain aforementioned techniques have also been evaluated in industrial settings. For instance, Oliveira et al. [33] assess the effect of employing different classifiers, resampling strategies, and sliding window sizes for issue assignment. Aktas and Yilmaz [34] also compare different techniques and further investigate how automated issue assignment compares with manual issue assignment in practice and how it is perceived by end users.

The approaches discussed in the previous paragraphs are effective in several different scenarios, however they also have important limitations. First of all, most approaches focus mainly on bug tracking systems by extracting textual data and are not tested on project management tasks/issues beyond the narrow scope of bugs [1, 2]. Concerning the utilization of textual data, several approaches are limited to vector space models and do not incorporate semantics, which can be more effective when handling text features [3, 4]. Instead, topic modeling approaches extract useful information from text, and are capable of capturing semantic relations between tasks and contributors [7, 8, 9, 10]. However, these approaches typically require parameter tuning, which is not always straightforward. For instance, one has to determine the number of topics before running the topic modeling technique. For the same reason, these approaches are usually applied and assessed only on few specific projects (i.e. in certain cases no more than three [7, 8, 9]) for which the algorithms have to be tuned specifically.

On a similar note, deep learning techniques based on word embeddings [24, 25, 26, 27] have also been shown to effectively extract semantics from text, provided that their hyperparameters are optimally defined. However, these algorithms may require large amounts of data, which may not always be available, while they also require excessive training time, especially if performing some grid search technique for their parameters. Furthermore, current research suggests that topic modeling techniques and word embeddings are different types of methods, which can both be efficient when employed for extracting topics and top terms from document collections [35, 36].

In this paper we design a methodology with the aim of overcoming the aforementioned limitations. Our methodology extracts information from a Jira server installation and is applicable on the broader automated

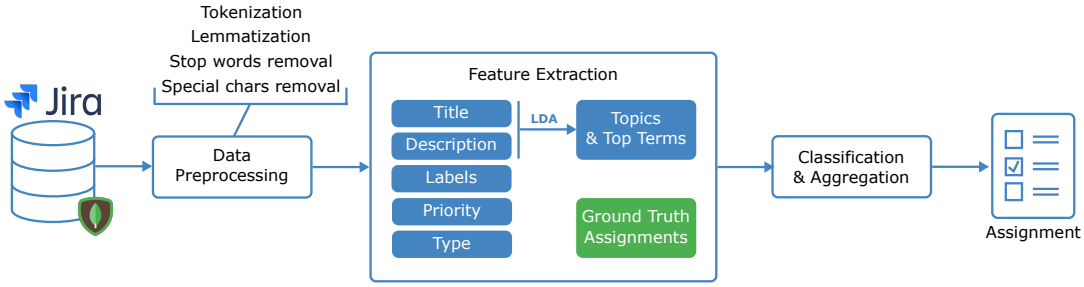

FIGURE 1 Overview of the System Architecture

task/issue assignment scenario. Furthermore, topic modeling is used to extract semantics from the textual data, while the optimization of our models is performed using the triaging results to boost its effectiveness. Finally, apart from textual data, our system further focuses on non-textual issue characteristics (labels, priorities, and types) and aggregates the results of multiple prediction models to recommend the most suitable developer for each new issue.

3 | METHODOLOGY

3.1 | Overview

Our methodology for automated issue assignment is outlined in Figure 1. As shown in this figure, our methodology comprises three steps: data preprocessing, feature extraction, and classification/aggregation.

The input is given in the form of the issue tracking data of a software project, which are initially filtered to remove missing/incomplete information, and then they are preprocessed to extract textual (title and description) and non-textual (labels, priority, type) features. The textual features undergo preprocessing and are placed on a vector space model so that they can be used for topic extraction using LDA. The results of the LDA are leveraged in two ways: as topic distributions, and as labels extracted from the top terms of topics. After that, different classification models are created for each individual feature, each producing issue assignment probabilities for the project contributors. And, finally, these probabilities are aggregated to produce the assignment.

Concerning the data used to build our approach, we extract the issue tracking data of the Jira infrastructure of the Apache Software Foundation[3]. The data are retrieved using version 2 of the Jira API and they are stored in a MongoDB instance, organized in six different collections: projects, users, issues, comments, events, and worklogs. Table 1 provides certain statistics about the dataset, which we made available online[4] to allow researchers in the field to replicate our study and/or build and evaluate new methods.

TABLE 1 Dataset Statistics

Metric	Value
Number of Projects	656
Number of Users	147610
Number of Issues	1013964
Number of Comments	4639882
Number of Events	7503864
Number of Worklogs	414893
Data Size	16.36GB (4.05GB compr.)

As the Apache Software Foundation includes a wide range of software projects that vary in size and complexity, our dataset is diverse, and covers different scenarios. An example issue of project Cassandra is shown in Figure 2. Notice that each issue comprises information not only about its title and description, but also about its

[3] https://issues.apache.org/jira
[4] https://zenodo.org/record/5665896

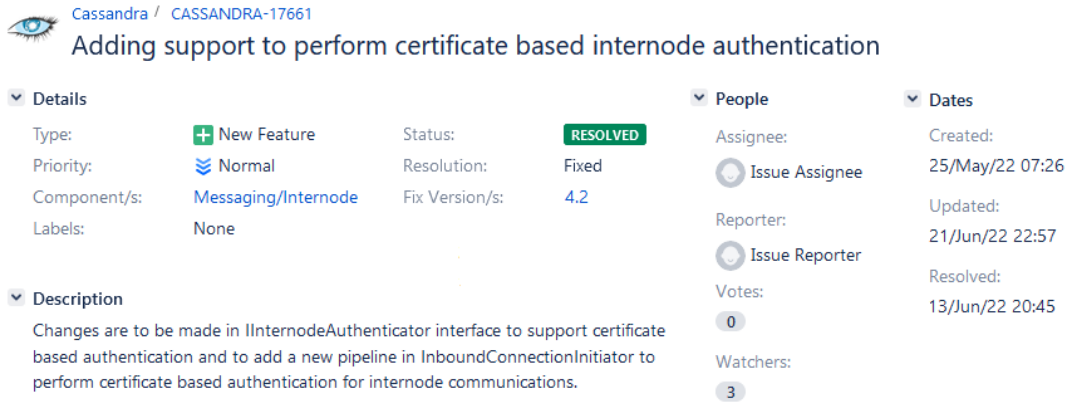

FIGURE 2 Example Issue of Project Cassandra

priority, its relevant components, its labels, etc. Furthermore, Jira issues have different types. In this case, the issue of Figure 2 describes a new feature to be developed for project Cassandra, which is relevant to certificate-based authentication.

The analysis is performed per project. To build an effective ground truth for issue assignment, we filter the dataset to obtain projects and issues that contain clear and sufficient information. In specific, we first filter out any issues without text data (title and description) as these cannot be used for our methodology. We further require that developers have a minimum number of 80 issues assigned to them (selected upon investigation), so that there is adequate set of assignments and developer history. Any developers with less involvement (external contributors) are dropped, along with the corresponding issues, while any projects with no developers or with fewer issues than this threshold are also dropped. After that, for each of the remaining projects, we perform the steps outlined in Figure 1, which are described in detail in the following subsections.

3.2 | Data Preprocessing

As already mentioned (see Figure 1), we use six issue fields in total for building our methodology: the title, the description, the labels, the priority, the type, and the assignee of each issue.

Concerning the text fields, these are arguably the ones containing the most important information of the issue, however they also are prone to much noise. As a result, we employ the spaCy [37] library to apply a series of preprocessing steps on the title and the description of each issue. First, we parse the text and remove all html and xml tags. Then, each text is split into tokens, discarding any punctuation and special characters. We also discard any stopwords found in the English stopwords list of spaCy, as well as any words with less than three characters. Finally, we convert all terms to lowercase and perform lemmatization to replace each word with its base form (e.g. 'best' becomes 'good'). As an example, the description of the issue of Figure 2 is transformed into the following tokens: [`change`, `interface`, `support`, `certificate`, `base`, `authentication`, `add`, `new`, `pipeline`, `add`, `perform`, `certificate`, `base`, `authentication`, `internode`, `communication`].

Concerning the labels of each issue, these are already split into clear terms/tokens, so no preprocessing was necessary for them. Concerning the non-textual data of the issues, these are replaced by ids, not only to allow the use of our algorithms but also in some cases to maintain consistency among different projects. In specific, issue priority is denoted in the scale Blocker-Critical-Major-Minor-Trivial in some projects and in the scale P0-P1-P2-P3-P4 in others. So, we replace both scales with integer ids (1, 2, 3, 4, 5). Concerning assignee

ids, the transformation is straightforward, as we can replace each individual by a unique id. Finally, the type of each issue has also to be replaced by an id according to Table 2.

TABLE 2 Conversion of Issue Type

#	Type	#	Type
1	Task	8	Quality Risk
2	Bug	9	Patch
3	Sub-task	10	Library Upgrade
4	Support Patch	11	Clarification
5	Feature Request	12	Epic
6	Enhancement	13	Tracker
7	Component Upgrade	14	Story

3.3 | Feature Extraction

As textual data are arguably quite important for issue assignment, we use two representation methods to capture both their lexical and their semantic features. The first representation is a vector space model that is used both as a feature in our methodology and as input to our topic modeling technique (second representation).

3.3.1 | Text Representation

We first build two simple text models, one for issue titles and one for issue descriptions. To do so, we employ a vector space model where texts (issue titles or issue descriptions) are represented as documents and words/terms are represented as dimensions. The vector representation for each text is created a Tf-Idf vectorizer, where the value of each term t in a text/document d is defined as:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{1}$$

In the above equation, the factor $tf(t, d)$ is the term frequency of term t in document d, and refers to the number of occurrences of term t in the document (issue title or issue description). The factor $idf(t, D)$ is the inverse document frequency of term t in the set of all documents D, and is used to penalize very common terms in the corpus (e.g. "issue" or "component"), as they may act as noise to our model. The $idf(t, D)$ is defined by the following equation:

$$idf(t, D) = log\left(\frac{|D|}{|d_t|}\right) \tag{2}$$

where $|d_t|$ is the number of documents (i.e. the number of issue titles or issue descriptions) containing the term t, and $|D|$ is the total number of documents in the corpus.

Finally, note that tf-idf is extracted using the training data and applied on the test data (more info about the splitting of data into training and test sets in subsection 4.1). As a result, as expected, any terms occurring only on test data will be ignored.

3.3.2 | Topic Modeling

Upon having tokenized and preprocessed the terms of issue titles and issue descriptions, we employ LDA [38] to extract a set topics that these documents are categorized into.

To extract maximum information from the text fields of each issue, we first concatenate the titles and the descriptions to produce one text (document) per issue. After that, if we consider that any document is a mixture of topics, LDA produces two probability distributions, one measuring the occurrence of terms in topics and one measuring the occurrence of topics in documents. Using the first distribution, we are able to extract the top words that describe each topic, while the second one provides the mixture of topics that comprise each document. LDA estimates both probabilities using a sparse Dirichlet prior distribution, denoted by the following equation:

$$D(\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} x_i^{\alpha_i - 1} \tag{3}$$

where K is the number of topics and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$

is a vector of the distribution parameters. The multivariate Beta function B is a normalizing constant, expressed using the Gamma function (Γ):

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)} \tag{4}$$

We used the Gensim [39] implementation of LDA, which is capable of learning the parameter vectors of the distributions directly from the data. The challenge, thus, is determining the number of topics. In our implementation, we optimize this parameter using the triaging result, connecting this step of our methodology to the next one, i.e. that of classification. Thus, for each project we iterate over the following numbers of topics: [4, 6, 8, 10, 12, 14, 16, 18, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170], selected for performance reasons. After that, for each configuration we extract the outputs of the LDA, which are the distribution of issues to the topics and the top terms.

After that, we now have different configurations (one per number of topics), thus we have to determine the optimal number of topics and select the relevant configuration. As a result, we perform the classification (issue assignment) for all configurations (i.e. for 4, 6, 8, etc. topics) and compute the accuracy for every configuration. Finally, we set the parameter for the number of topics to the one that produces the highest accuracy (see subsection 4.4 for an example optimization).

Upon having determined the number of topics, we can now forward the distribution of issues to the topics and the top terms for each topic in the classification step. As an example, in Figure 3 we provide an example of applying LDA for project Cassandra to extract 8 topics, visualized using the python port of LDAvis [40]. As shown in this figure, the topics seem to be rather balanced in this case. Concerning the top terms for topic 1, certain of them seem to be relevant to pdf documents (e.g. page, pdf, image, font, document) and others seem relevant to partitioning (e.g. partition, allocate, cleanup, buffer), indicating that it could be effective to re-run LDA with higher number of topics in order to produce more separated topics.

3.4 | Classification

Upon having constructed the different features for our analysis, we can now perform classification to assign issues to the most suitable project developers. The final recommendation is based on all features extracted, including the textual features (title and description), the non-textual features (labels, priority, and type), and the topic modeling features (issue-topic distribution and top terms). Specifically for the results of topic modeling we utilize the features in two ways. Concerning the issue-topic distribution, we use it as is, i.e. providing a vector of topic probabilities for each issue. Concerning the top terms, we first determine the dominant topic in the issue under analysis and then we extract its top terms and merge them with the issue labels. This way we produce what we may call *enhanced labels*. Thus, our final set of features includes the title, the description, the enhanced labels, the topics (issue-topic vector), and the other (priority and type).

After that, we train different classifiers for each feature, i.e. we train one classifier for the title, one for the description, etc. We evaluate two different sets of classifiers in our analysis, based on Naïve Bayes and SVM. Each of the classifiers provides as output a set of probabilities corresponding to different developers of the projects, indicating how probable is the assignment of the issue to each of them. As a final step, we aggregate all probabilities in the developer axis to produce a final assignment probability for each developer. Concerning the aggregation, we note that individual software teams may employ different ways of building software. Therefore, we introduce weights that are multiplied with the results of the classifiers, providing the final output as a weighting average. Of course, these weights are configurable and allow adapting to the specifics of each team. For the projects of the Apache dataset, our analysis shows that a configuration of [0.6, 0.7, 0.5, 0.5, 0.1] for the set of features [title, description, enhanced_labels, topics, other] is effective.

Thus, given an issue, these five models produce different probabilities for each assignee and the aggregated probability for each assignee is produced by their

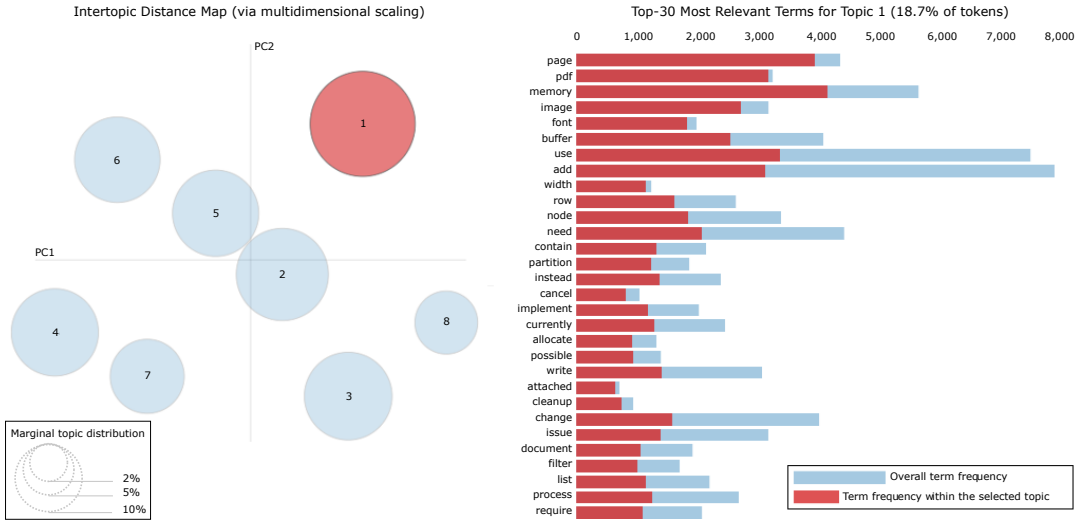

FIGURE 3 Example LDA Visualization for Project Cassandra

weighted average (where the weights are determined by the aforementioned configuration). For example, if the predicted probabilities of assigning a new issue to the first contributor of a project were [0.8, 0.4, 0.6, 0.2, 0.4], then the aggregated probability of assigning the issue to this contributor would be $(0.6 \cdot 0.8 + 0.7 \cdot 0.4 + 0.5 \cdot 0.6 + 0.5 \cdot 0.2 + 0.1 \cdot 0.4)/(0.6 + 0.7 + 0.5 + 0.5 + 0.1)$, which is equal to 0.5 (or 50%).

4 | EVALUATION

4.1 | Evaluation Framework

In this section, we evaluate our system using a set of four different experiments in order to approach different research questions. Our first experiment assesses the effect of including/excluding the various features outlined in the previous section. After that, for our second experiment, we investigate how the number of developers (and, thus, triaging choices) affects the performance of our models. The third and the fourth experiment evaluate the effectiveness of the LDA model, illustrating its performance when configured for different numbers of topics and assessing the effect of the enhanced labels, respectively. Finally, we perform one last experiment to compare our two possible classifiers, i.e. Naïve Bayes and SVM. Note that we use SVM as our main classifier for the first four experiments, as we found it to be more effective than Naïve Bayes.

Finally, in all experiments we use 80% of the issues as training set and 20% as test set. Cross-validation is performed at the training level for parameter optimization (i.e. it is used to determine the parameters of the SVM), while the dataset is split into training and test data before executing our methods to avoid introducing any bias on our evaluation.

4.2 | Evaluation on Different Projects

The main goal of this experiment is to assess the effectiveness of different feature configurations when performing issue assignment. As our methodology performs feature aggregation, we focus on the combination of features. As already mentioned, the features we use are: the title, the description, the enhanced labels (i.e. issue labels and top terms from topics), the topic distributions, and the other features (i.e. priority and type). Thus, the combinations we assess are: (i) the title on its own, (ii) the title and the description, (iii) the title, the description, and the enhanced labels, (iv) the title,

the description, and the topic distributions, and finally (v) all features (title, description, enhanced labels, topic distributions, other). This order of configurations was selected to assess how adding more and more information to our feature set can provide better results. For this experiment, we use a set of 12 projects with minimum 5 developers that have been assigned at least 80 issues.

The results of our analysis are shown in Table 3 (accuracy), Table 4 (F-measure) and Table 5 (AUC). As expected, the most effective approach is the one combining all possible features. An interesting observation is that the enrichment of labels with top terms seems to outperform the use of full topic distributions. This may be attributed to the fact that topic distributions are very descriptive, therefore the discriminative properties of the classification algorithms might require more data to output better results. Finally, concerning the overall performance of our approach, which, it reaches, on average, around 60% accuracy, which is quite effective, considering that there are 5 assignees per project.

TABLE 3 Accuracy of Different Configurations

Project	Title	Title & Desc.	Title & Desc. & Labels	Title & Desc. & Topics	All
AMBARI	0.51	0.57	0.61	0.59	**0.64**
ARROW	0.34	0.49	0.49	0.47	**0.50**
CASSANDRA	0.55	0.64	0.62	0.64	**0.65**
CB	0.41	0.45	0.50	0.45	**0.54**
DATALAB	0.45	0.54	0.65	0.54	**0.68**
FLINK	0.54	0.50	0.57	0.51	**0.56**
GEODE	0.31	0.42	**0.49**	0.42	0.44
HDDS	0.36	0.50	0.54	0.51	**0.56**
IGNITE	0.65	0.70	0.81	0.70	**0.85**
IMPALA	0.39	0.49	**0.60**	0.50	0.57
MESOS	0.44	0.59	**0.66**	0.57	0.61
OAK	0.34	0.49	0.53	0.49	**0.57**

TABLE 4 F-measure of Different Configurations

Project	Title	Title & Desc.	Title & Desc. & Labels	Title & Desc. & Topics	All
AMBARI	0.51	0.57	0.61	0.59	**0.64**
ARROW	0.34	0.49	0.49	0.48	**0.50**
CASSANDRA	0.55	0.64	0.62	0.64	**0.65**
CB	0.41	0.45	0.50	0.45	**0.54**
DATALAB	0.45	0.54	0.65	0.54	**0.68**
FLINK	0.54	0.50	**0.57**	0.51	0.56
GEODE	0.31	0.42	**0.49**	0.42	0.44
HDDS	0.36	0.50	0.54	0.51	**0.56**
IGNITE	0.65	0.70	0.81	0.70	**0.85**
IMPALA	0.39	0.49	**0.60**	0.50	0.57
MESOS	0.44	0.59	**0.66**	0.57	0.61
OAK	0.34	0.49	0.53	0.49	**0.57**

TABLE 5 AUC of Different Configurations

Project	Title	Title & Desc.	Title & Desc. & Labels	Title & Desc. & Topics	All
AMBARI	0.80	0.89	**0.92**	0.90	0.91
ARROW	0.69	0.73	0.72	0.73	**0.74**
CASSANDRA	0.81	0.84	0.88	0.85	**0.88**
CB	0.71	0.74	0.82	0.75	**0.82**
DATALAB	0.78	0.82	0.92	0.82	**0.93**
FLINK	0.80	0.82	0.85	0.82	**0.85**
GEODE	0.60	0.71	**0.72**	0.70	0.71
HDDS	0.72	0.79	**0.81**	0.80	0.80
IGNITE	0.89	0.93	**0.97**	0.93	0.97
IMPALA	0.71	0.80	**0.84**	0.80	0.84
MESOS	0.74	0.82	0.87	0.82	**0.88**
OAK	0.69	0.79	0.84	0.79	**0.85**

4.3 | Effect of Number of Assignees

The second experiment of our evaluation focuses on the impact of the different number of developers on the performance of our combined model. We use project FLINK for this evaluation as it has a large number of issues assigned to at least 20 assignees. The results of our analysis for 5, 10, 15, and 20 developers are shown in Figure 4 (accuracy) and Figure 5 (F-measure).

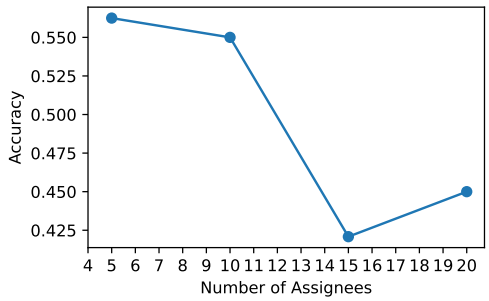

FIGURE 4 Task Assignment Accuracy for Different Number of Assignees

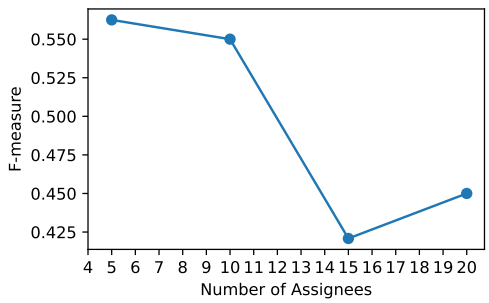

FIGURE 5 Task Assignment F-measure for Different Number of Assignees

As expected, the number of project developers is very important for the performance of our model. As the number of developers becomes higher, the performance of our model descends, however its effectiveness overall is quite high, considering the expected results of random assignment (e.g. for 10 developers, our model achieves accuracy equal to 0.54, whereas a random classifier would achieve around 0.10). Finally, concerning the small spike in our model metrics for 20 developers, this may depend on several factors, such as the specifics of each repository or the workload of individual developers.

4.4 | Optimizing the Number of Topics

As already mentioned in subsection 3.3.2, a significant shortcoming of LDA is that one has to provide the expected number of topics beforehand. To overcome this shortcoming and produce an optimized result, we determine the number of topics for each project using the accuracy of our final model. As an example, in Figure 6 we provide this analysis for the Cassandra project and for 5 developers. Several interesting conclusions can be made from this graph.

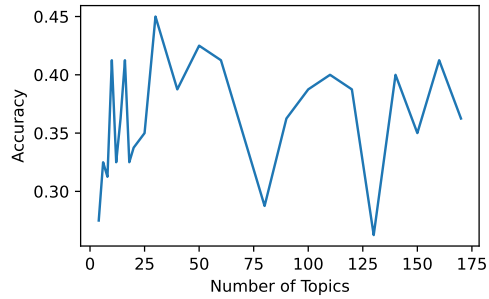

FIGURE 6 Task Assignment Accuracy for Different Number of Topics in LDA Technique

At first, one may note that the result of this analysis does not necessarily represent the best possible distribution of issues to topics. This is because the target of the optimization is not to perfectly assign the issues into topics; instead we focus on annotating the issues of the project (with a topic assignment and with relevant top terms) so that they are assigned to the most suitable developer. That said, the number of topics seems to exhibit some type of correlation with the accuracy of the model. Of course, as expected, very few topics with regard to the size of the project (e.g. less than 20 topics in this case) are not adequate to distinguish

among the semantics of different issues. On the other hand model performance seems to be rather unstable when the number of topics is above a threshold (e.g. 80 in this case), meaning that the semantic information is again not optimally extracted. All in all, the highest accuracy value for project Cassandra seems to be provided for 30 topics, so this is the value used from our system for this project.

4.5 | Effect of Enhanced Labels

Our next experiment evaluates the semantic information extracted by the LDA in a more straightforward way. To do so, we compare two individual configurations, a model that considers the labels of each issue versus a model that considers the enhanced labels that we have created, i.e. the ones merging the labels of the issue with the top terms of its dominant topic. The results of our analysis for the set of 12 projects are shown in Figure 7 (accuracy) and Figure 8 (F-measure).

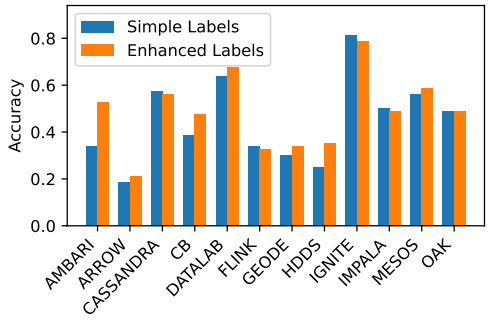

FIGURE 7 Task Assignment Accuracy for Simple versus Enhanced Labels

As shown in these figures, using the enhanced labels is more effective than using the simple ones for all projects. This difference is more obvious in the AMBARI, CB and HDDS projects, where the improvement by using enhanced labels is quite significant. Thus, we may safely assume that the top terms of the extracted topics contain useful information for our issue assignment task.

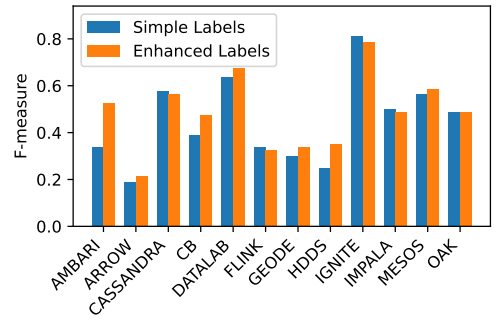

FIGURE 8 Task Assignment F-measure for Simple versus Enhanced Labels

4.6 | Assessment of Different Classifiers

The goal of our final experiment is to evaluate the effectiveness of different classifiers on the challenge of automated task assignment. We assess two rather popular classifiers in our analysis, a Naïve Bayes classifier and an SVM, given that they are highly preferred by researchers in this area, possibly because of their effective application on sparse features extracted from text. The results of the comparison are shown in Figure 9, for 12 projects with 5 assignees. SVM slightly outperforms Naïve Bayes considering the accuracy, the F-measure and the AUC metrics. This indicates that the descriptive ability of the current feature set has reached a high threshold; thus, in order to outperform these algorithms, we would have to proceed in different kinds of analyses.

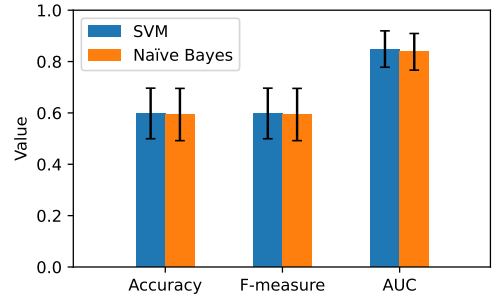

FIGURE 9 Task Assignment Accuracy, F-measure and AUC for Different Classifiers

5 | THREATS TO VALIDITY

The main threats to validity and limitations of our approach involve the choice of the evaluation dataset, the choice of the algorithms used, and the lack of comparison with other approaches. Concerning our selection of projects, we face limitations relevant to the existence of sufficient issue data. Given that text and especially topic modeling techniques require significant amounts of data for training, we had to omit projects with limited data as well as to drop issues with missing data (e.g. issues with no description). Additionally, we considered assignees with significant contribution in the project, so any secondary/external contributors are not taken into account for building our methodology. However, we may safely assume that these prerequisites are on par with current practice in certain software teams; core development in open-source projects is usually performed by few specific individuals (typically called *maintainers*), while external contributions may come from a large pool of different community members.

Regarding the different techniques used throughout this work, we employ Naïve Bayes and SVM as these two robust models are typically applied by current literature [1, 2, 4]; thus, a complete assessment of different models, including possibly ensembles or even deep learning approaches such as feature fusion [41], deviates from the scope of this paper. Our focus is on employing topic modeling, and specifically LDA, to extract semantic information from issues. Thus, we compare our methodology with a tf-idf baseline in order to determine whether the issue information is indeed semantically enriched and design our experiments to highlight the difference between vector space modeling and semantic information. To make this comparison focused, we do not employ word embedding techniques, such as word2vec [42] or fastText [43], and/or more recent transformer models (e.g. BERT [44]), which are also known for their effectiveness in these types of data [24, 25, 26, 27].

On a related note, comparing our technique, and generally topic modeling techniques, with word embeddings on an issue assignment scenario poses an interesting idea for future research. Both categories of algorithms have been shown to perform efficiently on typical topic extraction scenarios [35, 36], with each category encompassing different characteristics. Although methods like word2vec embed words in a latent vector space and thus allow us to extract word semantics and use analogies, LDA can inherently capture higher order of co-occurrences of terms (which is possible because topics are assumed to follow multinomial distributions over terms [38]) in an interpretable manner [45]. A rather interesting common ground is offered by Topic2Vec [46], a hybrid model that employs LDA to construct log-likelihood objective functions, which are subsequently maximized using word2vec (both CBOW and Skip-gram versions of the model). Thus, one future direction lies on applying this algorithm (or, possibly, the lda2vec [45] algorithm) on the issue assignment challenge, to further assess the effectiveness of topic modeling for issue assignment.

Concerning the comparison with other approaches in automated issue assignment, this was not directly possible, as most approaches use different types of datasets and thus they are optimized towards different goals. Specifically, most approaches focus mainly on bug triaging, which concerns the maintenance phase of the software development process, and thus extract data from bug databases. On the other hand, the Jira issues of the Apache Software Foundation are rather generic, as they are used for a variety of reasons, including feature development, discussions on enhancements, crafting of patches and/or upgrades, and even support. Moreover, they are sometimes used to communicate messages between the core maintainers of the project and the project users, a scenario certainly worth of further investigation. In any case, we plan to assess the applicability of our methodology with respect to current approaches, either by crafting suitable datasets or even by adapting our models. Finally, in an effort to allow other researchers to easily reproduce our findings, we have also uploaded our code online[5].

[5] https://github.com/AuthEceSoftEng/issue-assignment-with-topic-modeling

6 | CONCLUSION

Effective collaboration has lately grown to be an important research area for software engineering. In this paper, we have focused on the challenge of automated issue assignment, with the aim of saving considerable time and effort during the software development process. Our methodology uses diverse data from a Jira installation and builds a set of features that capture the semantics of different types of issues. Moreover, we employ a topic modeling technique that is optimized taking into account the final assignment, thus mitigating the need for manual tuning, while achieving significant improvement over simple vector space modeling. Finally, as supported also by our evaluation, we see that employing non-textual features further improves the results, and produces effective recommendations.

Concerning future work, we can improve several different parts of our methodology. First of all, we plan to create a more sophisticated model for aggregating different features, one that could automatically adapt to different projects, even outside the scope of our current dataset. Furthermore, we could investigate the incorporation of even more features, including e.g. issue comments or even components, that may offer valuable information when making an assignment. Finally, an interesting research direction would be to assess different types of text models (including e.g. word embeddings, such as word2vec [42], fastText [43], or even lda2vec [45]), with the aim of further enriching the semantics that can be extracted from issues, and subsequently improve the accuracy of the final assignment.

REFERENCES

[1] Murphy G, Cubranic D. Automatic Bug Triage using Text Categorization. In: Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering SEKE '04, USA: Knowledge Systems Institute; 2004. p. 92–97.

[2] Anvik J, Hiew L, Murphy GC. Who Should Fix This Bug? In: Proceedings of the 28th International Conference on Software Engineering ICSE '06, New York, NY, USA: Association for Computing Machinery; 2006. p. 361–370.

[3] Matsoukas V, Diamantopoulos T, Papamichail M, Symeonidis A. Towards Analyzing Contributions from Software Repositories to Optimize Issue Assignment. In: 2020 IEEE International Conference on Software Quality, Reliability and Security QRS 2020, Vilnius, Lithuania: IEEE Press; 2020. p. 243–253.

[4] Xuan J, Jiang H, Ren Z, Yan J, Luo Z. Automatic Bug Triage using Semi-Supervised Text Classification. In: Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering SEKE '10, USA: Knowledge Systems Institute; 2010. p. 209–214.

[5] Mani S, Sankaran A, Aralikatte R. DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging. In: Proceedings of the ACM India Joint International Conference on Data Science and Management of Data CoDS-COMAD '19, New York, NY, USA: Association for Computing Machinery; 2019. p. 171–179.

[6] Lee SR, Heo MJ, Lee CG, Kim M, Jeong G. Applying Deep Learning Based Automatic Bug Triager to Industrial Projects. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering ESEC/FSE 2017, New York, NY, USA: Association for Computing Machinery; 2017. p. 926–931.

[7] Ahsan SN, Ferzund J, Wotawa F. Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine. In: Proceedings of the 2009 Fourth International Conference on Software Engineering Advances ICSEA '09, USA: IEEE Computer Society; 2009. p. 216–221.

[8] Yang G, Zhang T, Lee B. Towards Semi-Automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-Feature of Bug Reports. In: Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference COMPSAC '14, USA: IEEE Computer Society; 2014. p. 97–106.

[9] Naguib H, Narayan N, Brügge B, Helal D. Bug Report Assignee Recommendation Using Activity Profiles. In: Proceedings of the 10th Working Conference on Mining Software Repositories MSR '13, IEEE Press; 2013. p. 22–30.

[10] Xia X, Lo D, Ding Y, Al-Kofahi JM, Nguyen TN, Wang X. Improving Automated Bug Triaging with Specialized Topic Model. IEEE Trans Softw Eng 2017;43(3):272–297.

[11] Papamichail MD, Diamantopoulos T, Matsoukas V, Athanasiadis C, Symeonidis AL. Towards Extracting the Role and Behavior of Contributors in Open-source Projects. In: Proceedings of the 14th International Conference on Software Technologies ICSOFT 2019, Prague, Czech Republic: SciTePress; 2019. p. 536–543.

[12] Greene GJ, Fischer B. CVExplorer: Identifying Candidate Developers by Mining and Exploring Their Open Source Contributions. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering ASE 2016, New York, NY, USA: Association for Computing Machinery; 2016. p. 804–809.

[13] Sharma M, Bedi P, Chaturvedi KK, Singh VB. Predicting the Priority of a Reported Bug using Machine Learning Techniques and Cross Project Validation. In: 2012 12th International Conference on Intelligent Systems Design and Applications ISDA 2012, IEEE Press; 2012. p. 539–545.

[14] Tian Y, Lo D, Xia X, Sun C. Automated Prediction of Bug Report Priority Using Multi-Factor Analysis. Empirical Softw Engg 2015;20(5):1354–1383.

[15] Kanwal J, Maqbool O. Bug Prioritization to Facilitate Bug Report Triage. Journal of Computer Science and Technology 2012;27(2):397–412.

[16] Alkandari MA, Al-Shammeri A. Enhancing the Process of Requirements Prioritization in Agile Software Development - A Proposed Model. J Softw 2017;12(6):439–453.

[17] Diamantopoulos T, Galegalidou C, Symeonidis AL. Software Task Importance Prediction based on Project Management Data. In: 16th International Conference on Software Technologies ICSOFT 2021, Held Online: SciTePress; 2021. p. 269–276.

[18] Lamkanfi A, Demeyer S, Giger E, Goethals B. Predicting the Severity of a Reported Bug. In: 2010 7th IEEE Working Conference on Mining Software Repositories MSR '10, IEEE Press; 2010. p. 1–10.

[19] Yang CZ, Hou CC, Kao WC, Chen IX. An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection. In: Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01 APSEC '12, USA: IEEE Computer Society; 2012. p. 240–249.

[20] Diamantopoulos T, Papamichail M, Karanikiotis T, Chatzidimitriou K, Symeonidis A. Employing Contribution and Quality Metrics for Quantifying the Software Development Process. In: Proceedings of the IEEE/ACM 17th International Conference on Mining Software Repositories MSR '20, Seoul, South Korea: ACM; 2020. p. 558–562.

[21] Alkhazi B, DiStasi A, Aljedaani W, Alrubaye H, Ye X, Mkaouer MW. Learning to rank developers for bug report assignment. Applied Soft Computing 2020;95:106667.

[22] Sajedi-Badashian A, Stroulia E. Vocabulary and time based bug-assignment: A recommender system for open-source projects. Software: Practice and Experience 2020;50(8):1539–1564.

[23] Li Z, Zhong H. Revisiting Textual Feature of Bug-Triage Approach. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering ASE '21, IEEE Press; 2022. p. 1183–1185.

[24] Guo S, Zhang X, Yang X, Chen R, Guo C, Li H, et al. Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network. Neural Process Lett 2020;51(3):2589–2606.

[25] He H, Yang S. Automatic Bug Triage Using Hierarchical Attention Networks. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C); 2021. p. 1043–1049.

[26] Wang H, Li Q. Effective Bug Triage Based on a Hybrid Neural Network. In: 2021 28th Asia-Pacific Software Engineering Conference (APSEC); 2021. p. 82–91.

[27] Lee J, Han K, Yu H. A Light Bug Triage Framework for Applying Large Pre-Trained Language Model. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering ASE '22, New York, NY, USA: Association for Computing Machinery; 2023. p. 1–11.

[28] Mohsin H, Shi C, Hao S, Jiang H. SPAN: A Self-Paced Association Augmentation and Node Embedding-Based Model for Software Bug Classification and Assignment. Know-Based Syst 2022;236(C).

[29] Jeong G, Kim S, Zimmermann T. Improving Bug Triage with Bug Tossing Graphs. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering ESEC/FSE '09, New York, NY, USA: Association for Computing Machinery; 2009. p. 111–120.

[30] Bhattacharya P, Neamtiu I. Fine-Grained Incremental Learning and Multi-Feature Tossing Graphs to Improve Bug Triaging. In: Proceedings of the 2010 IEEE International Conference on Software Maintenance ICSM '10, USA: IEEE Computer Society; 2010. p. 1–10.

[31] Xi SQ, Yao Y, Xiao XS, Xu F, Lv J. Bug Triaging Based on Tossing Sequence Modeling. J Comput Sci Technol 2019;34(5):942–956.

[32] Etemadi V, Bushehrian O, Akbari R, Robles G. A scheduling-driven approach to efficiently assign bug fixing tasks to developers. Journal of Systems and Software 2021;178:110967.

[33] Oliveira P, Andrade RMC, Barreto I, Nogueira TP, Morais Bueno L. Issue Auto-Assignment in Software Projects with Machine Learning Techniques. In: 2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP); 2021. p. 65–72.

[34] Aktas EU, Yilmaz C. Automated issue assignment: results and insights from an industrial case. Empirical Software Engineering 2020;25:3544–3589.

[35] Jędrzejowicz J, Zakrzewska M. Word Embeddings Versus LDA for Topic Assignment in Documents. In: Computational Collective Intelligence Cham: Springer International Publishing; 2017. p. 357–366.

[36] Sia S, Dalmia A, Mielke SJ. Tired of Topic Models? Clusters of Pretrained Word Embeddings Make for Fast and Good Topics too! In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP) Association for Computational Linguistics; 2020. p. 1728–1736.

[37] Honnibal M, Montani I. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing; 2017, to appear.

[38] Blei DM, Ng AY, Jordan MI. Latent Dirichlet Allocation. J Mach Learn Res 2003;3:993–1022.

[39] Rehurek R, Sojka P. Gensim–Python Framework for Vector Space Modelling. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic 2011;3(2).

[40] Sievert C, Shirley K. LDAvis: A Method for Visualizing and Interpreting Topics. In: Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces; 2014. p. 63–70.

[41] Liu Y, Qi X, Zhang J, Li H, Ge X, Ai J. Automatic Bug Triaging via Deep Reinforcement Learning. Applied Sciences 2022;12(7).

[42] Mikolov T, Chen K, Corrado G, Dean J. Efficient Estimation of Word Representations in Vector Space. In: Proceedings of the 1st International Conference on Learning Representations ICLR '13; 2013. .

[43] Joulin A, Grave E, Bojanowski P, Mikolov T. Bag of Tricks for Efficient Text Classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2 Valencia, Spain: Association for Computational Linguistics; 2017. p. 427–431.

[44] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. CoRR 2018;abs/1810.04805.

[45] Moody C. Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec. CoRR 2016;abs/1605.02019.

[46] Niu L, Dai X, Zhang J, Chen J. Topic2Vec: Learning distributed representations of topics. In: 2015 International Conference on Asian Language Processing (IALP); 2015. p. 193–196.