2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) | 978-1-6654-2705-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/SEAA53835.2021.00022

NLP4IP: Natural Language Processing-based Recommendation Approach for Issues Prioritization

1st Saad Shafiq
Johannes Kepler University
Linz, Austria
saad.shafiq@jku.at

2nd Atif Mashkoor
Johannes Kepler University
Linz, Austria
atif.mashkoor@jku.at

3rd Christoph Mayr-Dorn
Johannes Kepler University
Linz, Austria
christoph.mayr-dorn@jku.at

4th Alexander Egyed
Johannes Kepler University
Linz, Austria
alexander.egyed@jku.at

***Abstract*—This paper proposes a recommendation approach for issues (e.g., a story, a bug, or a task) prioritization based on natural language processing, called NLP4IP. The proposed semi-automatic approach takes into account the priority and story points attributes of existing issues defined by the project stakeholders and devises a recommendation model capable of dynamically predicting the rank of newly added or modified issues. NLP4IP was evaluated on 19 projects from 6 repositories employing the JIRA issue tracking software with a total of 29,698 issues. A comprehensive benchmark study was also conducted to compare the performance of various machine learning models. The results of the study showed an average top@3 accuracy of 81% and a mean squared error of 2.2 when evaluated on the validation set. The applicability of the proposed approach is demonstrated in the form of a JIRA plug-in illustrating predictions made by the newly developed machine learning model. The dataset has also been made publicly available in order to support other researchers working in this domain.**

***Index Terms*—Agile software development, natural language processing, issues prioritization**

I. Introduction

In agile project management platforms like JIRA[1] features (user stories), software enhancements, bugs, tasks, or other related activities are represented as issues. Product owners or managers may assign these issues with a story point, which denotes the effort required for their implementation [1]. Prioritization plays an essential role in ensuring that the most important issues are addressed in the early releases of a product. This also contributes towards high-quality products and better user acceptance [2]. The most important issues are identified through prioritization, which is mandatory to complete the release in a sprint. However, some studies, e.g., [3], [4], have shown a major void between the existing methodologies and the assumptions made in the agile software development literature regarding the role of end-users for the value creation process, the importance of business values for prioritization, and the role of the prioritization process in achieving project goals. The researchers in this area urge for the development of more prioritization approaches to bridge this gap.

The research reported in this article has been partly funded by the LIT Artificial Intelligence Lab and the LIT Secure & Correct Systems Lab supported by the state of Upper Austria.

[1]https://www.atlassian.com/software/jira

In this paper, we apply natural language processing (NLP) for issues prioritization. NLP has been successfully applied to requirements traceability, assessment, and task allocation in the past showing promising results [5]–[7]. We believe similar dividends can be harvested by exploiting NLP in the field of prioritization. We leverage the implementation of the word sequences proposed by Coyotl-Morales et al. [8] to understand the ontology of issues (user stories, tasks, or bugs) for each project. This implementation is commonly utilized in NLP to address classification problems in literature. In this fashion, we address specific challenges related to prioritization, such as handling a large number of issues, better integration of stakeholders' preferences, and an enormous amount of effort consumption in the prioritization process.

The main contribution of this paper is an NLP-based machine learning approach (NLP4IP) for generating an approximate rank function for newly added issues by incorporating priority and effort (measured in terms of story points) attributes specified previously by project stakeholders. The proposed approach assists project managers, team leads, lead developers, etc., in the decision-making process for sprint planning. The already provided priority attribute of existing issues alone is not sufficient for the prioritization of newly added issues; therefore, we supplement it with the effort to generate accurate recommendations. Unfortunately, both the priority and effort attributes contain arbitrary and highly subjective values. For example, a stakeholder may assign one issue in project A the priority value "critical". In contrast, another issue of similar priority can be ranked as "blocker" in project B by another stakeholder. We resolve this by grouping words of similar meaning and relative effort distribution for each project into categories discussed in Section III. In essence, our approach provides a harmonized ranking value: the priority attribute coupled with the associated implementation effort (calculated in terms of story points). Knowing the size of the issue (in terms of story points) beforehand lets the product owners better prioritize their releases [9]; thus, story points play a vital role in affecting the priority of incoming issues. The resulting rank value of issues – based on our approach – is aligned with the recommendation of Berander et al. [10], who suggest assigning requirements a numerical number based on an ordinal scale determining their priority. Moreover, the proposed approach can generate the approximate rank for

978-1-6654-2705-0/21/$31.00 ©2021 IEEE
DOI 10.1109/SEAA53835.2021.00022

issues dynamically over the entire course of project evolution. We also developed a prototype to demonstrate our approach's applicability and made the dataset employed in this study publicly available to the researchers.

To determine the efficacy of the proposed approach, we evaluate it across a large variety of projects that use the online software repository JIRA. We believe such repositories are a reliable and befitting source of obtaining textual issues and relevant attributes associated with them. Furthermore, the necessity of training an NLP model is to essentially have a large dataset, which could easily be extracted from JIRA. The obtained results showed an accuracy of 81% and a mean squared error (MSE) of 2.2 when evaluated on the dataset of all projects combined.

The rest of the paper is organized as follows: Section II discusses the related work. Section III presents the proposed approach. The NLP4IP algorithm is presented in Section IV. Section V demonstrates the implementation of the proposed approach through a benchmark study. Threats to the validity of the proposed approach are discussed in Section VI. Finally, the paper is concluded in Section VII.

II. Related work

In the past, a few studies have been conducted applying machine learning to requirements prioritization. For example, a technique known as CBRank proposed by Perini et al. [11] generates an approximation function from a set of previously elicited requirements from stakeholders. The technique is based on the RankBoost algorithm [11], which computes the approximation score by learning weak binary classifiers. The classifiers take elicited requirements and their corresponding stakeholder preferences (as weights) as input. Results obtained from the classifiers are compared with the initial weights, then the weights are readjusted iteratively to minimize the misclassification loss. The limitation of the approach is its limited applicability during evolution and continuous rank updates, i.e., the approach does not perform well when new requirements are being added or modified in a project.

Tonella et al. [12] used genetic algorithms to obtain users' knowledge regarding the importance of pair of requirements and mapping this information with the dependencies enlisted in requirement documents. The approach was evaluated on a real software system. The study results show that the proposed approach performs well when the number of elicited requirements is within the range of 25-125. Thus, an increase in cost and effort with the increased number of requirements tend to become a limitation of this study.

Kanwal et al. [13] proposed an approach to building a recommender system based on bug report features. Classifiers including support vector machine (SVM) and naive bayes (NB) were employed and compared in the study. The approach was evaluated on the Eclipse project with approx. 1,600 bug reports. Results show that NB outperforms SVM when trained on categorical features. However, SVM performs better when trained on textual features due to its ability to handling high-dimensional features. Nonetheless, a scarce number of bug reports and a limited number of models employed for comparison affect the generalizability of this study.

A method named EVOLVE proposed by Greer et al. [14] takes the stakeholders' priorities and constraints regarding efforts into account. It employs a genetic algorithm to find the best possible solution (a release plan) for the decision-makers to look up to while planning their release. The approach has been evaluated using a case study containing 20 requirements. However, the generalization of the proposed solution can not be evaluated due to the limited set of undertaken requirements.

McZara et al. [15] proposed a requirements prioritization method called "SNIPR" using NLP and constraint solvers. The NLP was utilized to identify dependent requirements by analyzing similarities in the textual features of these requirements. The proposed approach was evaluated empirically using a controlled experiment. In contrast, our approach uses NLP to vectorize the textual features in issues and uses them as input to the ML models.

As also observed by Achimugu et al. [16] and Bukhsh et al. [17], the aforementioned techniques suffer from issues such as excessive time consumption, continuous rank updates, incorporating live stakeholders feedback, dependencies among requirements, potentially fallible results, and lack of fully production-ready approaches for prioritization. In contrast to the aforementioned works, which take a set of predefined requirements as input and provide an ordered set of requirements as output, our work encompasses issues including user stories, tasks, and bugs. We take the newly added issues as input and automatically generate their ranks as output. Our work also addresses the aforementioned limitations by proposing a production-ready approach comprising an ML model capable of dynamically generating the predicted rank for an incoming issue, thus solving the issues of time consumption and continuous rank updates. As user stories in agile software development are not necessarily dependent on each other [18], dependencies among requirements are also not an issue here. Due to the absence of pairwise comparison (existing works use disagreement measures) and the derivation of the ranked dataset from projects, we use top@k accuracy and MSE as evaluation metrics. However, please note that a direct baseline comparison is not possible due to all these fundamental differences between our approach and other related works.

III. Approach

The NLP4IP approach provides a recommendation model that can predict an approximated rank whenever an existing issue is changed or a new issue is added to the project. In order to support the managers at the issue-reporting time, our approach takes the textual features of the issue, i.e., title and description, into account to gain the domain knowledge and ontology of the project, which ultimately help the machine learning model to understand the vocabulary specific to the project. Moreover, the priority and effort (measured in terms of story points) of issues provided by stakeholders are coupled together while computing the approximated rank in our

approach, which provides a harmonized ranking value and ensures stakeholders' prior preferences throughout the process.

We further analyzed our dataset to understand the most contributing feature towards "duration to resolution".[2] We then performed *RandomForest Regression* on the extracted features obtained from the change history of JIRA issues. Fig. 1 shows the regression results illustrating the top three important features for predicting duration to resolution. As anticipated, story points are the most contributing feature for predicting duration to resolution, affecting the priority, followed by the number of comments made on each issue and the number of links each issue has.

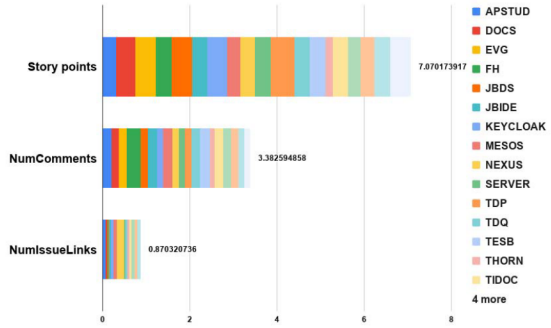

Fig. 1: Important features for duration to resolution

Please note that the approach NLP4IP only considered issues with previously assigned priority and story point attributes. The NLP4IP approach also caters to continuous rank updates by retraining the model after each sprint or iteration of the project. Furthermore, we have considered issues prioritization in agile software development as a multi-class classification problem to interlink textual issues with labeled classes. As illustrated in Fig. 2, NLP4IP comprises three main processes: the derivation of the ranked training dataset, the training process, and the validation process.

A. Derivation of the ranked training dataset

1) Issues categorization by priority: We measure the priority of each issue using the priority attribute of JIRA. The project stakeholders specify the values of the attribute before the beginning of each sprint. We followed the principles of the open coding method [19] during the analysis of the obtained dataset. The open coding method refers to analyzing the data, developing categories considering the features, and introducing labeling notions to build up a grounded theory. The dataset contains multiple values associated with the priority attribute, e.g., Critical, Major, or Optional. These values may vary from project to project. To overcome this, we have grouped the values with similar meaning (synonyms) into simple three generic classes encoded as class values $(0, 1, 2)$. Thereby, we have categorized the priority

[2]Duration to resolution is calculated as the difference between the issue creation time and the issue resolved time.

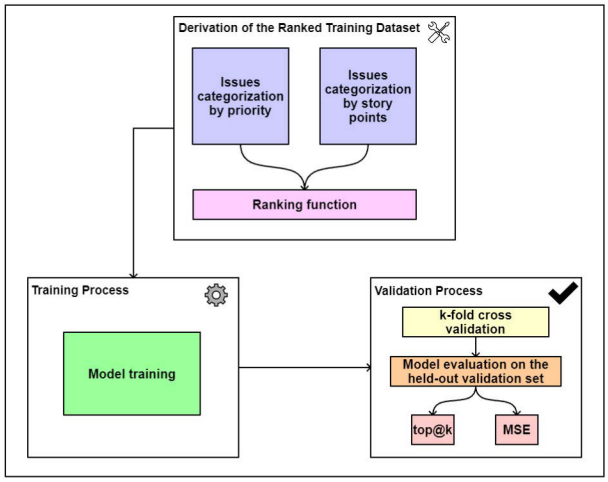

Fig. 2: Overview of the NLP4IP approach

of issues represented as $P(x)$, where $(2, 1, 0)$ represents $Classes(x) : High(x{=}Critical, Blocker, ...), Medium(x{=}Major, High, ...), Low(x{=}Optional, Low, ...)$, respectively.

$$P(x) = \begin{cases} 2 & if\ x = Critical, Blocker, ... \\ 1 & if\ x = Major, High, ... \\ 0 & if\ x = Optional, Low, ... \end{cases}$$

2) Issues categorization by story points: We measure the implementation effort of issues in terms of story points. It is essential for product owners and agile managers to know whether the issue can be included in the current sprint considering the time frame and workload. Thus story points help in calculating project velocity. This also enables managers to determine the issues that can be completed in a sprint; thus, prioritizing them would ensure early incremental deliverables. Story points are assigned to each issue by the stakeholders (development team) during the planning of each release [1], [20]. Similar to the issues categorization by priority, the observable values of story points are also divided into classes. The range value for each class is devised based on the sample distribution of the projects used as subjects in our case study. Therefore, these range values can vary from project to project. Consequently, we have categorized the effort (in terms of story points) of issues as $S(x)$, where $(0, 1, 2)$ represents $Classes(x)$: $Low, Medium, High$, respectively.

$$S(x) = \begin{cases} 0 & if\ x = Low \\ 1 & if\ x = Medium \\ 2 & if\ x = High \end{cases}$$

3) Ranking function: Then, we compute R – rank values – from $P(x)$ and $S(x)$. These rank values are assigned so that the issue with higher priority value and least effort required will be considered first and so on. This is based on the assumption that the dataset we acquired is pre-discussed and ultimately assigned after the project stakeholders' consensus. Hereby, rank values indicate the level of consideration for an

issue among others and can vary from range {1,2,3,...,9}, where least being the issue (ranked as 1) to be considered first. This concept of ranking has been inherited from the work of Berander et al. [10], where ranking is defined as a numerical assignment of issues based on an ordinal scale of 1 to *n* (1 being the highest priority). In order to achieve this, we take the Cartesian product of $P(x)$ and $S(x)$. In this way, we acquire a ranked training dataset for the training process.

$$P(x) \times S(x) = \begin{pmatrix} P^1 \\ P^2 \\ P^3 \end{pmatrix} \times \begin{pmatrix} S^1 \\ S^2 \\ S^3 \end{pmatrix}$$

$$R(x) = \begin{bmatrix} (P_1, S_1) \\ (P_1, S_2) \\ \vdots \\ (P_3, S_3) \end{bmatrix} = \begin{bmatrix} (2,0) \\ (2,1) \\ \vdots \\ (0,2) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 9 \end{bmatrix}$$

B. The training process

The training process comprises of training the ML model with specified hyper parameters.[3] The textual data contained in the ranked training dataset consists of the title and description of each issue, which is vectorized by tokenizing the text and converting it into a sequence of integers. Each integer represents the index of the token. These vector representations are later utilized as features in the training of the ML model. The rank values corresponding to the vector representations for each issue are used as labels in the training process. Note that our approach is independent of the training models being employed. Therefore, the results of the training models may vary from project to project.

C. The validation process

In the validation process, the performance of the ML model is evaluated by following standard ML practices in the literature. First, we employed the k-fold cross-validation process on our dataset to select the model having the best performance and avoid over-fitting. The cross-validation process is generally performed to attain a good assessment of the model prediction capability [21], [22]. It is well-understood that evaluating the model on the same data it was trained on would result in overoptimistic outcomes [23]. Therefore, the cross-validation process is meant to reduce this problem by splitting the data sample into K parts – commonly referred to as folds. Then, train the model on $K-1$ folds and test on the remaining fold. This process is continued K times. We further trained the selected model on the entire data once the assessment is complete and the model with the highest prediction capability (measured using the overall accuracy) is selected. Finally, we evaluated it on a held-out validation set for the corresponding project. We have also carried out the same process with all projects combined to understand the generalized aspect of our approach. In all the aforementioned steps, we ensured that the data is time-aware, i.e., we used the past historical data as our train set and the new data as our held-out validation set.

To evaluate NLP4IP, we have considered top@k accuracy and MSE as our primary evaluation metrics as they are commonly employed for multi-class classification problems [24]–[26]. These metrics are measured using True Positives (TP), True Negatives (TN), False Positives (FP) – also known as Type-I errors, and False Negatives (FN) – also referred to as Type-II errors. The employed metrics are explained below.

Top@k accuracy represents the percentage of how many predicted ranks are correctly recommended by the model within top k ranks. The *isTrue(n,k)* function returns "1" as output if at least one predicted rank is correct in the top k predictions made by the model; otherwise, "0" for each issue n in the set of issues N.

$$\textbf{top@k accuracy} = \frac{\sum_{n \in N} isTrue(n,k)}{|N|}$$

MSE represents the average error squares, i.e., it shows the difference between the true rank and the predicted rank. Mathematically, it can be written as the following equation where n denotes the number of issues, $True$ denotes the true rank, and $Pred$ represents the predicted rank.

$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(True - Pred)^2$$

IV. Algorithm

The NLP4IP approach takes in as input the issues, i.e., the concatenated title and description $Issue = \{issue_1....issue_n\}$, categories of priority (expressed as priority attribute) $P = \{P_1....P_n\}$, and categories of effort (expressed as story points attribute) $S = \{S_1....S_n\}$. As an output, the NLP4IP approach returns the approximate value obtained by the ranking function $R'(x)$ against any newly added or modified issue ($issue'$). R' thus represents the final predicted rank represented as $Classes(x)$: $R1 \sim R9$. We adopt the ranking scale of Massey et al. [27] due to the absence of pairwise comparisons of issues in our proposed approach.

We represent the NLP4IP approach as a pseudo-code in Algorithm 1. We also perform time-aware random sampling (H) on the dataset before splitting it into folds (K) to reduce the model's bias towards a particular class.

Each iteration (i) in the total number of folds (K) contains the following processes in the algorithm:

- Line [1-3] - Compute rank (R), generate random samples (H), and initialize the total number of folds (K)
- Line 4 - Start loop with the total number of iterations equals K

[3]Hyper parameters are configurations that can be modified in order to control the behavior of the machine learning model.

Algorithm 1 The NLP4IP approach

Input
Issue = {issue$_1$....issue$_n$}
Concatenated Title-Description per issue
P = {P$_1$....P$_n$}
Defined categories by Priorities per issue
S = {S$_1$....S$_n$}
Defined categories by Story Points per issue
Output

$$R'(x) = [R' : issue \implies issue']$$

Begin

1: R = ComputeRank(P,S)
Compute rank by creating sets of Category P and Category S
2: H = RandomSampling(Issue,R)
A random sampling of the dataset based on computed R
3: K = 5 [Total number of folds (train/test sets)]
Divide the sample into 5 folds
4: **for** Each fold i in K **do**
Equalize train set classes
5: T$_i$ = TrainModel(H)
Train model with specified hyperparameters
6: V$_i$ = Validate(T$_i$)
7: **if** predicted rank = true rank **then**
8: TruePredictions + 1
9: **end if**
10: TotalPredictions + 1
11: Compute Precision, Recall, F$_1$-Score per class
12: Compute the average Accuracy and MSE for i
13: **end for**
14: M = Trained model with high V$_i$ accuracy
15:

$$R'(x) = \sum_{n=1}^{n} M_n(issue')$$

16: return $R'(x)$

End

- Line 5 - Train (T) ML model on the train set for a given fold i
- Line 6 - Validate (V) trained model on the test set for a given fold i
- Line [7-10] - Compute number of true predicted values by comparing them with true values in the test set
- Line 11 - Compute the Precision, Recall and F$_1$-Score measure for each class
- Line 12 - Compute the average Accuracy and MSE for a given fold i
- Line 13 - End loop
- Line [14-16] - Return the approximate rank value generated by the selected model corresponding to the newly added or modified issue ($issue'$)

V. Benchmark study

We performed a benchmark study comprising 19 different projects obtained from six JIRA repositories to evaluate our approach. The projects were selected based on predefined project selection criteria explained in the following subsection.

A. Preliminaries

There is a pipeline of processes that are prerequisites for this study. First, we ensure through the definition of criteria that only relevant projects evolved over a certain period of time are included in the study. Then, we employ common machine learning processes, such as data extraction and data preprocessing, which are necessary to build a high-performance machine learning model.

1) Project selection criteria: We have selected projects from six major repositories using the JIRA issue tracking software. These repositories include: apache[4], appcelerator[5], jboss[6], sonatype[7], talendforge[8], and mongodb[9]. Initially, 44 projects were selected based on our criterion C1 (listed below), which were then further scrutinized. A total of 19 projects were eventually included in the final pool. Overall, this study contains 29,698 issues. Following is the project selection criteria:

C1: The project must have assigned priority and story points attributes.
Rationale: Both of these attributes are necessary for the NLP4IP approach as they demonstrate the priority and effort required to complete the issues. Therefore, we are only considering issues assigned with both of these attributes beforehand.

C2: The project must have a minimum of 400 issues and 10 sprints.
Rationale: We are interested in only those projects, which have a significantly large number of issues and evolving over a longer period of time. We fix a threshold value for issues and sprints as 400 and 10, respectively.

C3: The primary language used in the project must be English.
Rationale: We are using the English language corpora in our study; thus, we opted for projects having primary language as English.

C4: The project type must be “Software".
Rationale: We are not interested in projects other than the type Software such as Business or Service Desk.

2) Data extraction process: To acquire relevant data, we have designed a data extraction process tailored to the scope of this study. The data extraction process starts with using the REST API provided by the JIRA issue tracking software. The REST API provides endpoints for extracting the desired data attributes within a project. We have developed a custom

[4]https://issues.apache.org/jira
[5]https://jira.appcelerator.org/
[6]https://issues.jboss.org/
[7]https://issues.sonatype.org/
[8]https://jira.talendforge.org/
[9]https://jira.mongodb.org/

application in C# to extract the issues and their specified attributes for a particular project. The data is then transformed into the required format. An example of an issue (user story) containing the attributes extracted from the projects is shown in Fig. 3.

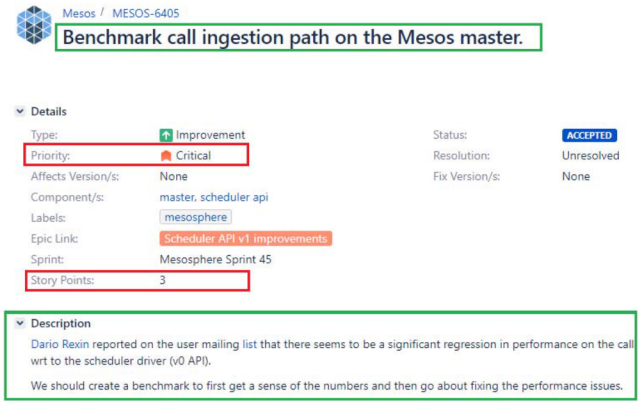

Fig. 3: An example of attributes extracted from each issue, where $Title$ and $Description$ are highlighted in green and $Priority$ and $Storypoints$ in red.

3) Data preprocessing: We have ensured the data quality of our dataset by performing data preprocessing, which comprised removing the unnecessary words – called stop words – from the title and description of each issue using the English stop words corpus provided by NLTK (Natural Language Toolkit).[10] The removal of stop words is essential to increase the performance of the model and to ensure that only those words are included, which helps to identify the context of a sentence [28]. Moreover, we have also removed various HTML tags from the text for similar reasons. The final dataset has been made publicly available and described more in detail in Section V-F.

B. Study design

To compare the proposed approach, the benchmark study is comprised of 9 different ML models. This includes 1) Decision trees (DT), 2) K-nearest neighbour (KNN), 3) Support vector machine (SVM), 4) Logistic regression (LR), 5) Naive bayes (NB), 6) Random forest (RF), 7) XGBoost, 8) FastText, and 9) Long short-term memory (LSTM). Note that we chose these models as they have been widely used to address text classification problems and have shown notable performance in the past [13], [29]–[31].

As a part of the benchmark study, Algorithm 1 was implemented in python scripts (one for each ML model) and executed by providing the dataset for each project individually and then combined. To elaborate, we first distinctly evaluated each model on the single corresponding project's held-out validation set. Then, to illustrate the ability of NLP4IP on unseen projects, we evaluated NLP4IP in a cross-project setting where data samples from $n-1$ projects have been employed as training data while leaving data samples of 1 project out as testing data. This resulted in 19 iterations of the training and validation process, evaluating each project one by one. Lastly, to further determine our approach's generalizability, we trained the model based on the data accumulated collectively from all projects while randomly separating this held-out validation set and performed the evaluation on the combined and randomized held-out validation set.

While splitting the dataset between the train set and the validation set, we ensured that the data is time-aware, i.e., training on past issues and validation on newer ones. Before the training, we also set the story point range values (explained in Section III-A2) for each project based on the sample issues distribution. For instance, we categorized the middle majority samples as *Medium* and the rest as *Low* and *High* for a project with the bell-curve distribution. We kept the lower majority samples as *Medium* and the rest as *Low* and *High* for a project with exponential distribution.

C. Results

The results obtained after the learning process are twofold. First, we report on the MSE values of each model acquired from performing k-fold cross-validation on the dataset. Second, we report the benchmark study results where NLP4IP is applied in an individual project setting and a cross-project setting. Fig. 4 shows the boxplot illustrating the k-fold cross-validation results (in terms of MSE) of the ML models on the combined dataset employed in this study. SVM emerged as the highest performing model with an MSE of 3.7802 (std=2.1323), whereas NB performed the least with an MSE of 22.9444 (std=1.4351).

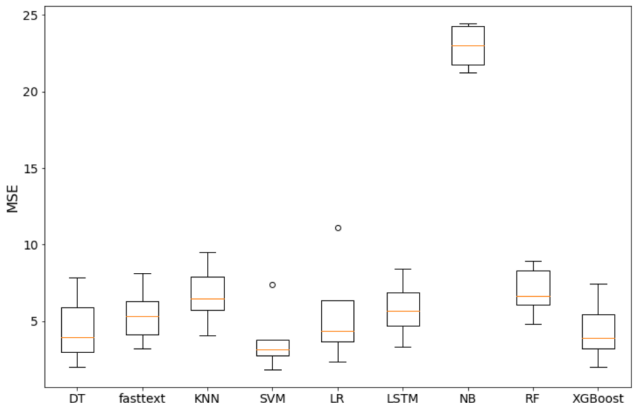

Fig. 4: k-fold cross-validation results (low is better)

Table I reports the results of NLP4IP when applied in an individual project setting. The first column "Project", reports the name of the projects employed under study. The rest of the columns report the top@3 accuracy and MSE values of the respective ML models. For each project, the entries shown in bold fonts indicate the best MSE values (highlighted in green) with the difference of 0.1 and the best top@3 values (highlighted in blue) with the difference of 0.01.

[10]https://www.nltk.org

TABLE I: Results of the validation process (individual project setting)

Project	DT top@3 | MSE	Fasttext top@3 | MSE	KNN top@3 | MSE	SVM top@3 | MSE	LR top@3 | MSE	LSTM top@3 | MSE	NB top@3 | MSE	RF top@3 | MSE	XGBoost top@3 | MSE
APSTUD	**0.62** | 17.66	0.42 | 12.02	0.49 | 13.66	0.46 | 15.49	0.40 | 14.46	0.44 | 13.50	0.44 | **7.65**	0.41 | 14.83	0.50 | 14.25
DOCS	0.66 | 3.10	0.66 | 3.29	0.72 | 3.18	0.73 | 3.13	0.69 | 5.36	**0.79** | 4.62	0.21 | 11.06	0.73 | 3.58	0.77 | **2.91**
EVG	**0.97** | **0.72**	0.81 | 0.89	0.91 | 0.78	**0.98** | **0.80**	**0.97** | 1.29	0.87 | 0.93	0.15 | 2.77	**0.98** | 1.11	**0.98** | **0.80**
FH	0.80 | 2.21	0.80 | 1.82	0.80 | 2.27	**0.92** | 2.55	**0.91** | 2.66	0.82 | **1.50**	0.12 | 15.79	**0.91** | 2.50	**0.92** | 2.51
JBDS	0.69 | 2.68	**0.79** | 3.23	0.64 | 3.83	0.72 | 3.27	0.58 | 5.01	0.64 | 5.22	0.12 | 12.82	0.73 | **2.41**	0.68 | 2.52
JBIDE	0.69 | 3.38	**0.77** | 4.01	0.51 | 3.91	0.67 | 3.31	0.59 | 6.34	0.70 | 3.36	0.15 | 16.46	0.69 | 3.85	0.67 | **3.07**
KEYCLOAK	0.79 | 3.21	0.78 | 3.84	0.70 | 3.73	0.76 | 3.91	0.69 | 4.49	**0.82** | **2.41**	0.17 | 5.28	0.76 | 3.36	0.78 | 3.32
MESOS	0.73 | **2.91**	**0.75** | 3.27	0.62 | 3.13	**0.74** | 3.14	0.66 | 5.03	0.70 | 3.41	0.15 | 12.73	**0.75** | 3.16	0.73 | **3.00**
NEXUS	0.57 | 5.11	**0.74** | **3.64**	0.47 | 4.15	0.56 | 5.0	0.55 | 6.22	0.64 | 3.81	0.31 | 9.62	0.64 | 3.78	0.61 | **3.66**
SERVER	0.91 | 1.70	0.75 | **1.35**	0.84 | 1.70	0.96 | 2.25	0.87 | 2.95	0.80 | 1.49	0.10 | 7.36	**0.98** | 1.83	**0.98** | 1.71
TDP	0.58 | **4.59**	**0.74** | 5.18	0.50 | 9.11	0.59 | 5.20	0.52 | 9.79	0.51 | 5.90	0.30 | 29.02	0.56 | 9.11	0.59 | 4.96
TDQ	0.49 | 8.39	**0.72** | 9.33	0.47 | 9.37	0.58 | 9.68	0.51 | 10.17	0.58 | **6.60**	0.32 | 13.19	0.49 | 9.87	0.57 | 10.37
TESB	0.48 | 5.29	**0.71** | **4.07**	0.39 | 5.24	0.52 | 4.76	0.45 | 5.88	0.59 | 5.14	0.15 | 13.35	0.45 | 4.92	0.51 | 5.08
THORN	0.69 | 3.03	0.71 | 3.60	0.65 | 3.78	0.67 | 4.05	0.54 | 6.33	0.57 | 4.04	0.16 | 4.99	0.69 | **2.77**	**0.73** | **2.85**
TIDOC	**0.70** | 7.99	**0.71** | **5.96**	0.49 | 7.10	0.62 | 7.87	0.51 | 8.59	0.69 | 8.03	0.13 | 36.97	0.62 | 7.62	**0.70** | 7.15
TIMOB	0.57 | 9.32	**0.68** | 10.71	0.43 | 10.66	0.56 | 9.85	0.50 | 11.27	0.59 | **9.03**	0.27 | 27.22	0.56 | 11.08	0.57 | 10.37
TISTUD	**0.73** | 5.06	0.68 | 6.67	0.63 | 5.73	0.76 | **4.86**	0.67 | 7.49	0.66 | 7.30	0.16 | 32.99	**0.74** | 5.55	**0.74** | **4.95**
WINDUP	0.74 | 2.90	0.68 | **1.55**	0.62 | 2.49	0.86 | 2.37	0.70 | 5.45	0.61 | 2.76	0.06 | 9.96	0.86 | 1.97	**0.90** | 1.75
WT	0.83 | 1.60	0.69 | **1.58**	0.75 | 1.71	0.86 | **1.49**	0.78 | 2.72	0.80 | 2.04	0.32 | 9.87	**0.87** | 1.75	**0.86** | **1.53**

Top model(s) (**MSE** = ~ 0.1, **top@3** = ~ 0.01) for each project emphasized

The results show that XGBoost appeared to be the best-performing model. It has performed better on 8 projects (considering both top@3 and MSE metrics) compared to the rest of the models. FastText (second best), on the other hand, has dominated individual projects (in terms of top@3 metric) due to its efficient learning of word representations by converting textual features into character n-grams. DT and LSTM are following in terms of performance with a slight difference in their accuracy. Apparently, we can see that NB has shown the least performance implying its inability to capture features from an imbalanced dataset.

As further can be seen in Table I, there are two projects (SERVER and EVG) with the highest top@3 accuracy and lowest MSE, indicating the fact that the model has successfully predicted the values due to the training on majority samples of classes falling under rank 4, 5, and 6. This implies that the unequal label distribution has a significant impact on the model's bias, which, in turn, results in fewer class predictions on most of the samples. Table II shows the results obtained by applying NLP4IP in a cross-project setting. The evaluation results for each project are based on the aforementioned $n-1$ strategy. These results demonstrate that SVM appeared to be dominant over the rest of the models in terms of the lowest MSE values. Overall, the XGBoost model performed with an accuracy of 81% with an MSE of 2.52 (slightly higher than SVM) when evaluated on a dataset of all projects combined. However, due to the imbalanced nature of the dataset, XGBoost appears to be the most befitting model as it provides a range of parameters to fine-tune the model on a given dataset. Moreover, the risk of overfitting associated with the XGBoost model was overcome by performing k-fold cross-validation elaborated in the aforementioned Section III-C.

D. Discussion and implications

The practical implication of NLP4IP is discussed in this section. Based on the results obtained from the individual project setting, the reported MSE values for each model indicate how far the predicted rank is from the true rank. For instance, in this study, the MSE value of 2.5 for a project implies that if the true rank of an issue is "4", then on average, the predicted rank could vary from 3-5. Thus, the lower the MSE value is, the better performance a model has. However, for some projects (APSTUD, TDQ, TIMOB, and TISTUD), we observed considerably high MSE values (> 6.0), which indicate that the disparity among the ranks recommended by the models and the true ranks is high. To better understand the cause, we performed K-means clustering to group issues with similar textual features and observed their variance. Apparently, the issues with the same cluster had different classes, also known as class overlap, which resulted in the worse performance of models on these projects. Further studies are required to investigate the potential solutions to overcome this problem.

The results obtained from the cross-project setting indicate that SVM outperformed other models except for XGBoost, which had slightly higher MSE but better accuracy. We also trained and validated the model on the entire dataset (combining all projects), which showed the performance of SVM and XGBoost on par with each other but below the per-project learning results. We observed that the text does not correlate with the effort and priority attributes as they might be project progress/situation dependent, which, in turn, suggests that cross-project learning should be used only when there is insufficient data available in a project to conduct per-project learning.

Summarizing results, among all the ML models employed in this study, XGBoost was able to capture the features more accurately and outperformed the rest due to its ensemble framework based on decision trees. In comparison to DT – the third-best – XGBoost complements decision trees with a gradient boosting algorithm to strengthen the prediction power of the model further. SVM follows XGBoost in terms of

TABLE II: Results of the validation process (cross-project setting)

Project	DT top@3 | MSE	Fasttext top@3 | MSE	KNN top@3 | MSE	SVM top@3 | MSE	LR top@3 | MSE	LSTM top@3 | MSE	NB top@3 | MSE	RF top@3 | MSE	XGBoost top@3 | MSE
APSTUD	0.34 | 8.03	**0.43** | 11.53	0.30 | 10.13	0.29 | **7.56**	0.30 | 8.10	0.37 | 10.47	0.28 | 21.89	0.34 | 10.25	0.33 | 7.85
DOCS	**0.70** | 2.91	0.48 | 5.47	0.46 | 5.59	**0.70** | **2.69**	0.64 | 3.83	0.54 | 5.56	0.13 | 19.87	0.45 | 6.55	0.68 | 3.60
EVG	**0.69** | 3.02	0.57 | 3.17	0.48 | 5.70	0.47 | **2.87**	0.64 | 3.05	0.59 | 3.73	0.04 | 12.84	0.56 | 4.32	**0.68** | 3.00
FH	0.84 | 3.74	0.57 | 5.05	0.56 | 4.87	**0.90** | **2.76**	0.86 | **2.67**	0.61 | 5.31	0.34 | 16.30	0.69 | 4.86	0.83 | **2.78**
JBDS	0.71 | 3.33	0.57 | 6.08	0.49 | 5.70	**0.74** | **3.06**	0.70 | 3.74	0.60 | 6.10	0.19 | 22.20	0.60 | 6.11	**0.73** | 3.51
JBIDE	0.72 | 3.17	0.57 | 5.17	0.46 | 5.64	**0.74** | **2.77**	0.70 | 3.65	0.57 | 6.27	0.26 | 19.47	0.55 | 6.51	0.68 | 3.62
KEYCLOAK	**0.80** | 2.53	0.58 | 4.57	0.53 | 4.95	**0.81** | **2.26**	0.78 | 2.63	0.64 | 5.34	0.18 | 19.24	0.64 | 5.24	**0.81** | 2.81
MESOS	0.77 | **2.84**	0.58 | 4.32	0.50 | 5.37	**0.79** | **2.78**	0.73 | 3.36	0.55 | 5.47	0.45 | 17.75	0.59 | 6.00	0.75 | 3.25
NEXUS	0.61 | 5.06	0.58 | 4.98	0.46 | 7.18	**0.67** | **4.19**	0.59 | 5.11	0.55 | 5.74	0.32 | 13.25	0.47 | 7.59	0.61 | **4.08**
SERVER	0.87 | 2.37	0.59 | 2.78	0.56 | 4.35	**0.96** | **2.15**	0.85 | 2.56	0.76 | 2.68	0.37 | 15.02	0.60 | 5.51	0.88 | **2.23**
TDP	0.30 | 9.75	**0.58** | 11.17	0.38 | 11.92	0.31 | 9.55	0.31 | 9.03	0.40 | 8.94	0.41 | 12.20	0.33 | 11.21	0.31 | **7.89**
TDQ	0.32 | 9.34	**0.56** | 8.85	0.40 | 11.26	0.33 | 8.74	0.32 | 8.73	0.45 | 8.59	0.41 | 11.73	0.36 | 10.39	0.34 | **7.75**
TESB	0.53 | 7.20	**0.56** | **6.55**	0.47 | 8.61	**0.56** | 7.04	0.51 | 6.72	0.50 | 6.74	0.48 | 11.22	0.45 | 9.05	0.51 | **6.57**
THORN	0.67 | 4.40	0.56 | 5.48	0.47 | 5.94	**0.73** | **3.56**	0.66 | 4.20	0.52 | 6.65	0.36 | 18.13	0.50 | 6.93	0.69 | **3.62**
TIDOC	0.36 | 6.07	**0.56** | 9.19	0.30 | 8.20	0.31 | **5.86**	0.32 | 7.61	0.49 | 10.17	0.15 | 33.29	0.38 | 9.11	0.41 | 7.01
TIMOB	0.24 | 5.90	**0.55** | 9.01	0.26 | 8.55	0.22 | **5.71**	0.26 | 8.20	0.43 | 10.50	0.21 | 28.48	0.39 | 8.93	0.28 | 7.15
TISTUD	0.30 | 5.28	**0.54** | 9.81	0.24 | 8.23	0.26 | **5.09**	0.27 | 6.99	0.46 | 10.63	0.13 | 34.50	0.37 | 10.23	0.33 | 7.07
WINDUP	0.80 | 3.77	0.54 | 5.13	0.55 | 5.66	**0.84** | 3.36	0.81 | **3.08**	0.59 | 4.48	0.33 | 15.77	0.68 | 5.08	0.80 | **3.11**
WT	0.64 | 3.77	0.55 | 3.43	0.51 | 6.70	0.87 | 3.85	0.68 | 4.05	0.64 | 4.31	0.50 | 6.96	0.52 | 5.78	**0.78** | **3.06**
ALL	0.76 | 2.90	0.71 | 3.70	0.63 | 5.08	0.73 | **2.23**	0.77 | 2.96	0.72 | 5.34	0.18 | 21.32	0.70 | 5.73	**0.81** | 2.52

Top model(s) (**MSE** = ~ 0.1, **top@3** = ~ 0.01) for each project emphasized

lowest MSE, which explains its high applicability in multi-class text classification. Since a specific architecture for the LSTM model is followed in this study, we can further conclude that despite the low performance of LSTM in a cross-project setting, further fine-tuning of the model and better model architecture could yield a significant increase in its performance.

NLP4IP can be utilized as a pre-assigned single criterion ranking technique. The ability to easily deriving the ranked dataset from the pre-assigned attributes within projects facilitates learning, thus improving the performance of the prioritization process. It also makes the NLP4IP domain-independent and suitable for any project with pre-assigned priority and effort. The results obtained from the benchmark study suggest that NLP4IP performs better when applied in an individual project setting. To implement NLP4IP on a new project, it is preferred that the project has a significant number of issues and the textual features of issues are distinguishable. On the contrary, if only limited data is available, cross-project learning could yield better results.

The presented benchmark study helps researchers analyze the comparable performance of the models, especially in large-sized projects. Moreover, our approach – NLP4IP – allows practitioners to get early warnings on newly added or modified issues by generating an approximated rank, which helps them prioritize issues more effectively. For instance, an early generated rank such as "1" against a corresponding newly created issue would indicate that the issue is expected to have high priority and low effort. Knowing this beforehand helps practitioners decide whether the issue should be considered in the upcoming sprint, thus alerting them on time.

E. Prototype implementation

To demonstrate the applicability of our approach, we have developed the *Rank indicator* plug-in as a recommendation tool for JIRA. The plug-in forecasts the rank of every newly added or modified issue. The source code of the prototype is publicly available at Github[11] and its outcome (highlighted in the green box) is shown in Fig. 5. The plug-in consumes the trained model to predict the rank of incoming issues and is readily available for exploitation.

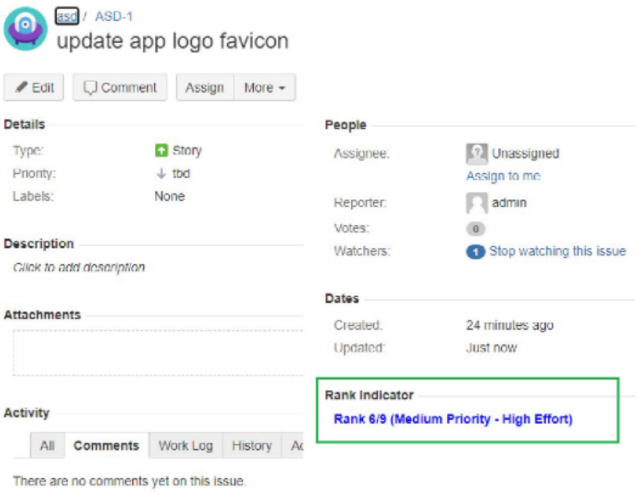

Fig. 5: Plug-in screenshot

F. Publicly available dataset

One of the novels and major contributions of this paper is the provision of a gold standard dataset[12] to facilitate the undertaking of further research on applied NLP techniques for issues or requirements prioritization. The provided dataset contains two folders naming "withChangeHistory" and "withoutChangeHistory". Both folders contain

[11]https://github.com/jku-isse/RankIndicator
[12]https://github.com/jku-isse/RankIndicator/tree/master/Dataset

TABLE III: Dataset attributes

Attribute Type	Attributes	Description
Categorical	ProjectName	Project's name
	Key	JIRA issue key
	Type	Issue Type
	Created	Issue Creation date
	LastUpdatedStatus	Last updated status
	LastUpdatedStatusDate	Date of last updated status
	Priority	Priority for each issue
Numeric	Storypoint	Story points
	NumWatchers	Number of watchers
	NumComments	Number of comments
	NumIssueLinks	Number of links to other issues
	NumAffectedVersions	Number of affected versions
	NumChangeHistory	Number of changes made for each issue
	NumFixVersions	Number of fixed versions
Textual	title_desc	Concatenated title and description of each issue
Derived	P, S, R	Derived columns from Section III-A

`.csv` files for each project. These files comprise categorical, textual, and derived attributes. For example, the folder named "withoutChangeHistory" contains `.csv` files for each project, excluding the attributes obtained from the change history of issues, i.e., `ProjectName`, `Key`, `title_desc`, `Created`, `Priority`, `Storypoint`, `P`, `S`, `R`. Table III describes the dataset attributes.

We further plotted a histogram showing the derived rank corresponding to the issues to understand the distribution better. Fig. 6 shows the rank distribution across the number of samples in the dataset. As can be seen, due to the imbalanced nature of the dataset, we meticulously split our dataset into training and validation sets. As a result, we maintained a similar ratio of classes among the sets when deriving ranked training data before the training process for each project.

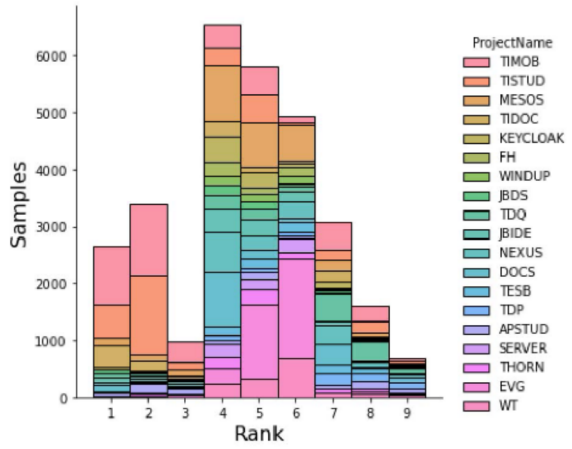

Fig. 6: Dataset rank distribution

A large corpus of dataset like this could help researchers as a valuable resource to conduct further research on issues prioritization and selection using other natural language processing or deep learning techniques. Also, the provided dataset could be instrumental in the area of cost and effort estimation in agile software development, e.g., as shown by Choetkiertikul et al. [20], who performed story points estimation using a deep learning model trained on textual requirements. The provided dataset could also be beneficial in building priority-based recommendation systems for incoming issues.

Further implications of the dataset could be strengthening the knowledge building process, addressing the challenge of building domain ontology models using textual requirements [5] and understanding the "template sentence structure" of the issues that specifically belong to the agile software development. Moreover, it can also be immensely beneficial in emotional software engineering and sentiment analysis studies [32], where we try to evaluate the emotional impact (presence/absence of emotions in text) of issues on priority and story points. This ultimately leads to the evaluation of project velocity and possibly allows project managers to better orient their issues or requirements elicitation process.

VI. Threats to validity

A. External validity

The external validity threat can influence the generalizability of results due to the insufficient amount of data employed in a study. In order to overcome this external threat, we have gathered a relatively large dataset comprising of 19 projects obtained from 6 different repositories. Moreover, the k-fold cross-validation process also supports the conclusion of the generality of our model in similar contexts.

B. Internal validity

The internal validity threat can occur due to a poor research methodology or a spurious apprehension of the research protocol. In order to overcome this threat, we have developed a custom application that automatically extracts and formats the required data acquired from the Rest API provided by JIRA. Furthermore, the dataset that we have used to evaluate our approach is entirely based on real projects that evolved over a certain time period, thus reducing any internal threat to the validity of this study.

C. Construct validity

This study is prone to construct validity as we are augmenting the priority attribute with story points in order to prioritize issues. However, as aforementioned, story points can be utilized as a metric to measure effort and as a rule of thumb, an issue with a priority value x with an effort value y is prioritized over the issue with a priority value x and an effort value y', where $y' > y$. Thus we believe that the two attributes (priority and story points) utilized in our approach are adequate to prioritize issues.

VII. Conclusion

In this paper, we have presented the NLP4IP approach for issues prioritization based on natural language processing. NLP4IP allows project managers to better orient their

sprint planning ahead of time by knowing the approximate rank for each newly added or modified issue. The use of NLP4IP is recommended for projects with readily available sufficient data. However, in case of data scarcity, cross-project learning can be employed. The ranking function of NLP4IP is devised by acquiring the priority attribute in combination with the unit of measuring effort (in terms of story points) for each issue given by the stakeholders. Moreover, we also conducted a state-of-the-art benchmark study to compare the performance of various ML models, which shows that NLP4IP can predict ranks for unseen issues with an average top@3 accuracy of 81% and MSE of 2.5 with the XGBoost model. Lastly, we made the dataset publicly available for the research community.

In the future, we intend to investigate the relevance and closeness of rank values within the k prediction(s) made by the model. For instance, a prediction of *(k=3)* "7, 8, 9" should be more relevant than a prediction of "1, 5, 9". In addition, we plan to employ pre-trained models and sentence embeddings from modern models such as BERT to further improve the capturing of domain knowledge and ontology of the model.

References

[1] M. Cohn, *Agile Estimating and Planning*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[2] M. Daneva and A. Herrmann, "Requirements Prioritization Based on Benefit and Cost Prediction : A Method Classification Framework," in *2008 34th Euromicro Conference Software Engineering and Advanced Applications*. Parma: IEEE, 2008, pp. 240–247.

[3] Z. Racheva, M. Daneva, K. Sikkel, and A. Herrmann, "Do we Know Enough about Requirements Prioritization in Agile Projects : Insights from a Case Study," in *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 147–156.

[4] M. Daneva, E. V. D. Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar, N. Ajmeri, U. Ramteerthkar, and R. Wieringa, "The Journal of Systems and Software Agile requirements prioritization in large-scale outsourced system projects : An empirical study," *The Journal of Systems & Software*, vol. 86, no. 5, pp. 1333–1353, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.12.046

[5] M. A. A. Andres Arellano, Edward Carney, "Natural Language Processing of Textual Requirements," *Proceedings of the Workshops, 19th International Conference Conference on Automated Software Engineering*, vol. 9, no. 3, pp. 93–97, 2015.

[6] R. Sharma, J. Bhatia, and K. K. Biswas, "Machine learning for constituency test of coordinating conjunctions in requirements specifications," in *International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. ACM, 2014, pp. 25–31.

[7] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, and A. Egyed, "TaskAllocator: A Recommendation Approach for Role-based Tasks Allocation in Agile Software Development," in *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 2021, pp. 39–49.

[8] R. M. Coyotl-morales and L. Villaseñor-pineda, "Authorship Attribution Using Word Sequences," in *Iberoamerican Congress on Pattern Recognition*. Berlin: Springer Berlin Heidelberg, 2006, pp. 844–853.

[9] K. Logue, K. Mcdaid, and D. Greer, "Allowing for Task Uncertainties and Dependencies in Agile Release Planning," in *Proceedings Software Measurement European Forum (SMEF)*, 2007, pp. 275–284.

[10] P. Berander and A. Andrews, *Requirements Prioritization*. Berlin: Springer, Berlin, Heidelberg, 2005.

[11] A. Perini, A. Susi, and P. Avesani, "A Machine Learning Approach to Software Requirements Prioritization," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 445–461, apr 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6249686/

[12] P. Tonella, A. Susi, and F. Palma, "Interactive requirements prioritization using a genetic algorithm," *Information and Software Technology*, vol. 55, no. 1, pp. 173–187, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2012.07.003

[13] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," *Journal of Computer Science and Technology*, vol. 27, no. 2, pp. 397–412, 2012.

[14] D. Greer and G. Ruhe, "Software release planning: An evolutionary and iterative approach," *Information and Software Technology*, vol. 46, no. 4, pp. 243–253, 2004.

[15] J. McZara, S. Sarkani, T. Holzer, and T. Eveleigh, "Software requirements prioritization and selection using linguistic tools and constraint solvers—a controlled experiment," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1721–1761, 2015.

[16] P. Achimugu, A. Selamat, R. Ibrahim, and M. N. R. Mahrin, "A systematic literature review of software requirements prioritization research," *Information and Software Technology*, vol. 56, no. 6, pp. 568–585, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2014.02.001

[17] F. A. Bukhsh, Z. A. Bukhsh, and M. Daneva, "A systematic literature review on requirement prioritization techniques and their empirical evaluation," *Computer Standards and Interfaces*, vol. 69, no. November 2019, p. 103389, 2020. [Online]. Available: https://doi.org/10.1016/j.csi.2019.103389

[18] M. Trkman, J. Mendling, and M. Krisper, "Using business process models to better understand the dependencies among user stories," *Information and Software Technology*, vol. 71, pp. 58–76, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2015.10.006

[19] S. H. Khandkar, "Open coding," University of Calgary, Tech. Rep., 2009.

[20] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, 2019.

[21] S. Geisser, "The Predictive Sample Reuse Method with Applications," *Journal of the American statistical Association*, vol. 70, no. 350, pp. 320–328, 1975.

[22] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," *Journal of the Royal Statistical Society*, vol. 36, no. 2, pp. 111–147, 1973.

[23] S. C. Larson, "The shrinkage of the coefficient of multiple correlation." *Journal of Educational Psychology*, vol. 22, no. 1, pp. 45–55, 1931.

[24] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2009.03.002

[25] Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration," *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, vol. 1, pp. 335–342, 2014.

[26] J. Jiang, Y. Yang, J. He, X. Blanc, and L. Zhang, "Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development," *Information and Software Technology*, vol. 84, pp. 48–62, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2016.10.006

[27] A. K. Massey, P. N. Otto, L. J. Hayward, and A. I. Anto, "Evaluating existing security and privacy requirements for legal compliance," *Requirements engineering*, vol. 15, no. 1, pp. 119–137, 2010.

[28] K. V. Ghag and K. Shah, "Comparative analysis of effect of stopwords removal on sentiment classification," in *IEEE International Conference on Computer Communication and Control, IC4 2015*. IEEE, 2016, pp. 2–7.

[29] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use?" in *Proceedings - 2013 12th International Conference on Machine Learning and Applications, ICMLA 2013*, vol. 2. IEEE, 2013, pp. 112–116.

[30] I. Scholtes, M. S. Zanetti, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: A case study on four open source software communities," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 1032–1041.

[31] R. Sepahvand, R. Akbari, and S. Hashemi, "Predicting the bug fixing time using word embedding and deep long short term memories," *IET Software*, vol. 14, no. 3, pp. 203–212, 2020.

[32] N. Novielli, "Sentiment and Emotion in Software Engineering," *IEEE Software*, vol. 36, no. August, pp. 6–23, 2019.