Software Task Importance Prediction based on Project Management Data

Themistoklis Diamantopoulos, Christiana Galegalidou and Andreas L. Symeonidis
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, Thessaloniki, Greece

Keywords: Task Importance, Bug Severity, Ordinal Classification, Project Management, Task Management.

Abstract: With the help of project management tools and code hosting facilities, software development has been transformed into an easy-to-decentralize business. However, determining the importance of tasks within a software engineering process in order to better prioritize and act on has always been an interesting challenge. Although several approaches on bug severity/priority prediction exist, the challenge of task importance prediction has not been sufficiently addressed in current research. Most approaches do not consider the meta-data and the temporal characteristics of the data, while they also do not take into account the ordinal characteristics of the importance/severity variable. In this work, we analyze the challenge of task importance prediction and propose a prototype methodology that extracts both textual (titles, descriptions) and meta-data (type, assignee) characteristics from tasks and employs a sliding window technique to model their time frame. After that, we evaluate three different prediction methods, a multi-class classifier, a regression algorithm, and an ordinal classification technique, in order to assess which model is the most effective for encompassing the relative ordering between different importance values. The results of our evaluation are promising, leaving room for future research.

1 INTRODUCTION

Software development nowadays is a collaborative process, taking place in online code hosting facilities, such as GitHub[1], and being supported by project management systems, such as Jira[2]. Using these types of tools, developers can monitor the software development process in a fine-grained way, by assigning tasks, prioritizing features, resolving bugs, planning releases, and generally keeping track of the evolution of their project.

In this collaborative context, and especially for large projects with multiple contributors, it is often required to prioritize the various tasks of a project. The process of determining the importance of a task (or an issue) is crucial, as it affects the prioritization of the task with respect to the development sprints as well as the expected duration for the completion of the issue itself. And yet the decision is usually non-trivial; there are typically no clear rules on how the level of importance is defined, thus leaving the decision up to the personal judgment of the team members. Consequently, the tasks of the project at hand are often flagged with inconsistent or badly planned importance values, this way complicating the management of the project and giving rise to major issues in project development.

We argue that a challenge that arises in this context is whether we can automate the process of task importance assignment in task management systems. Most contemporary research efforts have not focused specifically on this challenge; they have focused instead on the challenges of bug priority prediction (Sharma et al., 2012; Tian et al., 2015; Kanwal and Maqbool, 2012) and bug severity prediction (Lamkanfi et al., 2010; Lamkanfi et al., 2011; Yang et al., 2012; Roy and Rossi, 2014; Menzies and Marcus, 2008; Tian et al., 2012; Yang et al., 2014). Furthermore, the input of certain approaches is limited to textual data (i.e. issue titles and descriptions) and the time frame of the data is not considered. This way, the prediction may be based on data that are quite different from the issue under analysis, especially when they are extracted from issues more than a few months old. Finally, most approaches do not account for the ordinal characteristics of the output, i.e. the fact that the priority/severity of an issue is defined on an arbitrary scale where the relative ordering between different values is significant.

In this paper, we propose a methodology that over-

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

Diamantopoulos, T., Galegalidou, C. and Symeonidis, A.
Software Task Importance Prediction based on Project Management Data.
DOI: 10.5220/0010578302690276
In *Proceedings of the 16th International Conference on Software Technologies (ICSOFT 2021)*, pages 269-276
ISBN: 978-989-758-523-4

comes the aforementioned limitations. Given input from the task management system of a project, our system employs information including both textual and meta-data, such as the type or the assignee of the task, in order to predict the importance of a task. The time frame of the tasks is taken into account so that task priorities are determined by data that are temporally close. As far as the ordinal characteristics of the output are concerned, we apply three different models, a multi-class classifier, a regression model and an ordinal classifier in order to examine which method is better suited for the challenge at hand.

The rest of this paper is organized as follows. Section 2 disambiguates the term *importance* from the terms *priority* and *severity* and reviews the related work. Our methodology for a task importance recommender is presented in Section 3. Section 4 evaluates our approach against a set of software projects and Section 5 discusses the perceived challenges in the area of task importance prediction. Finally, Section 6 concludes this work and provides useful insight for future research.

2 RELATED WORK

As already mentioned, related work in the area of task importance prediction/classification is rather limited. There is, however, a considerable volume of work regarding bug priority prediction and bug severity prediction, which are similar challenges, yet require some careful disambiguation. In specific, we may define the priority of a task as *a measure that indicates how urgent it is to deliver that specific task.* On the other hand, severity determines *the impact of a task (or, much more often, a bug) on the project at hand* (Lamkanfi et al., 2011). In other words, priority answer to the question 'what to implement/fix first', whereas severity answers to the question 'how much impact does fixing this have to the system'.

In the context of this work, task importance can be defined as *a measure of how much the delivery of a task impacts the software project at hand.* Therefore, it is most similar to the concept of severity rather than that of priority[3]. Indeed, bug priority is usually measured in categories P1-P4 (or, sometimes, P1-P5), where P1 indicates that the bug must be fixed as soon as possible and P4 (or P5) indicates that the bug will never be fixed (Uddin et al., 2017). Severity, on the other hand, usually takes values such as Trivial, Minor, Major, Critical, and Blocker in increasing order of importance.

[3]Interestingly, however, we could note that defining the importance/severity of an issue can be quite helpful for prioritizing.

Further concerning severity prediction, which is most relevant to this work, certain research efforts consider a more coarse-grained output merging severity values into two levels of importance (i.e. severe and non-severe). One such approach is proposed by Lamkanfi et al. (2010) who used data from three open-source projects monitored by the bug tracking system Bugzilla[4]. The authors applied tokenization on bug descriptions and employed a Naïve Bayes classifier to classify bugs into severe and non-severe. Further extending their work (Lamkanfi et al., 2011), the authors evaluated three more algorithms, an 1-Nearest Neighbor classifier, a Support Vector Machines classifier, and a Naïve Bayes Multinomial classifier, proving the latter to be the most effective.

A similar approach was followed by Yang et al. (2012) who focused mainly on feature selection. The authors employed three different features selection techniques, the Information Gain, the chi-squared (χ^2), and the Correlation Coefficient, in order to determine the terms that are the most decisive for classifying bug reports in the severe and non-severe categories. Finally, another approach that focuses mainly on text algorithms is that of Roy and Rossi (2014). The authors also extract data from Bugzilla and evaluate three configurations, one with single text tokenization, one with tokens and bigrams, and one with tokens, bigrams, and chi-squared feature selection. Their results indicate the latter to be the most effective in predicting bug severity.

There are also several approaches that consider fine-grained output with five or more classes of severity. An example system in this category is SEVERIS (Menzies and Marcus, 2008), which is trained using data from the NASA PITS database. The system extracts tokens from bug reports and applies the Information Gain for feature selection before finally employing the RIPPER rule learner (Cohen and Singer, 1999) to determine the severity of each report. INSPect (Tian et al., 2012) follows a rather different approach, extracting both text tokens and other features (such as, e.g., the component that is relevant to the report) and creating a metric that computes the similarity between two Bugzilla bug reports. After that, INSPect employs a nearest-neighbors algorithm to determine the class of a bug report base on the class of its neighbors. Finally, another similar approach is proposed by Yang et al. (2014) who first employ LDA to categorize the bug reports to different topics and then consider the nearest neighbors of each bug report within its topic in order to predict its severity.

Although the aforementioned approaches are ef-

[4]https://www.bugzilla.org/

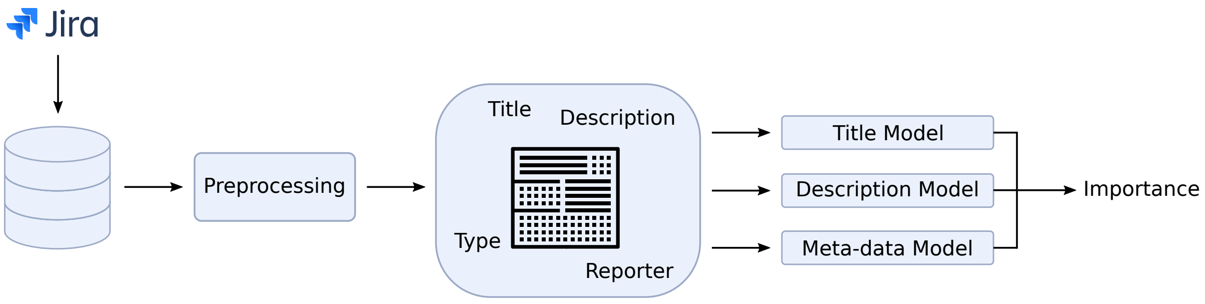


Figure 1: Methodology for Task Importance Assignment.

fective for bug severity prediction, they do not always conform to the challenge of task importance prediction. Tasks from project management systems may describe features, requirements, or even issues in the code or the documentation, whereas bugs refer usually to faults in the project. Thus, effectively predicting the importance of tasks requires considering the textual sources of information and modeling also the meta-data relevant to the type and the specifics of the task. In this paper, we propose a methodology that considers both types of information to predict the importance of tasks using models that are effective for ordinal characteristics.

3 METHODOLOGY

Our methodology for automated task importance assignment is outlined in Figure 1. The input is given in the form of the project management data of a software project, which is initially preprocessed to extract its text, description, and other information/meta-data (type, assignee). After that, we employ a different model for each type of (title, description, information) and, finally, we aggregate the outputs of all models in order to recommend the importance value.

Concerning the data, we have retrieved the dataset crafted by Ortu et al. (2015) and set up a PostgreSQL database instance according to the authors' instructions[5]. The dataset includes project management data extracted from the Jira repositories of the Apache Software Foundation[6]. It includes approximately 1000 projects with more than 700,000 tasks/issues. Our methodology is obviously agnostic, as it can receive input from any project management platform, however we use here data from Jira as a proof of concept. An example task for project CouchDB is shown in Figure 2.

Notice that each task comprises a handful of information, including not only its title and description, but also its status, its relevant components, its assignee, etc. Furthermore, at the right side of Figure 2, we may see the temporal characteristics of the task. Concerning importance, Jira actually defines it using the term 'priority', which however is rather inaccurate as discussed in Section 2. Despite this inaccuracy, this metric actually represents importance (or severity) as it receives the values *Trivial*, *Minor*, *Major*, *Critical*, and *Blocker*. Hence, we hereafter refer to this metric as 'importance'.

In our case, the analysis is performed per project, thus we have used a database connector to retrieve one project at a time and perform the steps outlined in Figure 1 to build a prediction model per project.

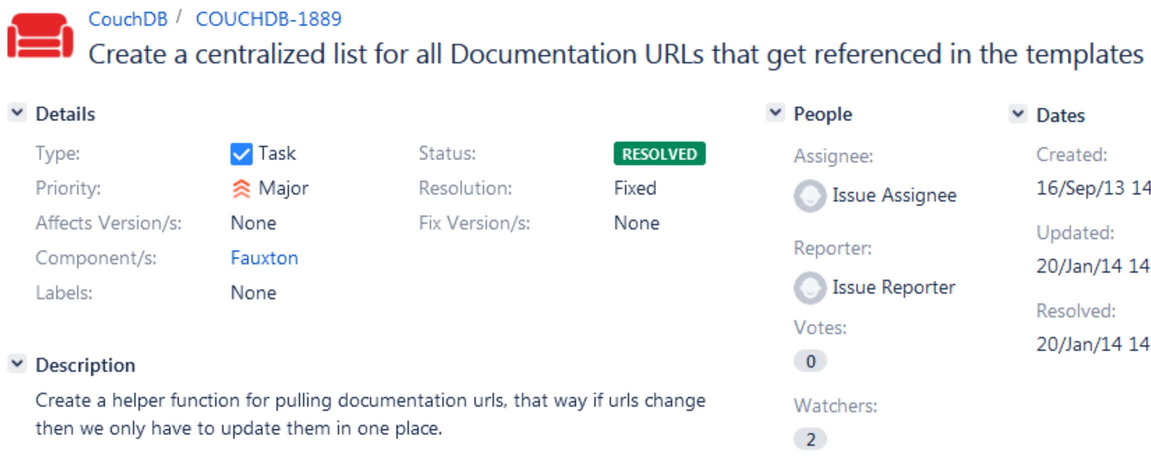
3.1 Data Preprocessing

As our methodology is applied on actual projects, several tasks may have incomplete or default importance values. Thus, the first step is to remove any tasks without importance value as well as any task with the default value given by Jira (i.e. 'Major') when the user selects not to assign an importance value.

After that, we preprocess the text fields of the tasks, i.e. the titles and the descriptions. Initially, we parse the texts and remove all html tags. Also, each text is split into tokens and all punctuation is removed. Then, we perform stemming to remove word endings (e.g. 'running' becomes 'run') and lemmatization to replace each term with its base form (e.g. 'better' becomes 'good'). The next step is to remove all numbers, single characters, and stop words using the list of NLTK[7]. As an example application of our preprocessing pipeline, the description of the task of Figure 2 is transformed into the following tokens: [create, helper, function, pull, document, url, way, url, change, update, one, place].

Finally, concerning the rest of the fields shown in Figure 2, we use the type and assignee, as these have been shown to yield better results in our analysis. Both fields are converted to ids. Assignees were

[5]https://github.com/marcoortu/jira-social-repository
[6]https://issues.apache.org/jira/
[7]https://www.nltk.org/


CouchDB / COUCHDB-1889
Create a centralized list for all Documentation URLs that get referenced in the templates

Details

Type:	Task	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	None
Component/s:	Fauxton		
Labels:	None		

Description

Create a helper function for pulling documentation urls, that way if urls change then we only have to update them in one place.

People

Assignee: Issue Assignee
Reporter: Issue Reporter
Votes: 0
Watchers: 2

Dates

Created: 16/Sep/13 14:33
Updated: 20/Jan/14 14:37
Resolved: 20/Jan/14 14:37

Figure 2: Example Task of Project CouchDB.

replaced by their corresponding system IDs, while the type of the task was replaced by a numerical value according to Table 1. For each project, we also flagged the types with less than 10 occurrences as outliers and removed the corresponding tasks from the project.

Table 1: ID Conversion of Task Type.

#	Type	#	Type
1	Bug	9	Story
2	Component Upgrade	10	Sub-task
3	Documentation	11	Support
4	Improvement	12	Task
5	New Feature	13	Temp
6	Quality Risk	14	Test
7	Question	15	Wish
8	Refactoring		

3.2 Data Models

As already mentioned, our methodology takes into account both the textual features and the meta-data of the tasks. Thus, in the following subsections we describe two data models, one used for the title and the description of the tasks and one used for their meta-data (type and assignees).

Task Text Model. We build two text models, one for task titles and one for task descriptions. For each one of them, we employ a vector space model where texts (titles or descriptions) are represented as documents and words/terms are represented as dimensions. Using the Tf-Idf vectorizer, we create the vector representation for each document/text. In specific, the weight (value of the vector) of each term t in a document d is defined as:

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D) \quad (1)$$

where the factor $tf(t,d)$ is the term frequency of term t in document d and refers to the number of occurrences of the term t in the document (title or description). The factor $idf(t,D)$ is the inverse document frequency of term t in the set of all documents (titles or descriptions) D. The $idf(t,D)$ in our implementation is defined as:

$$idf(t,D) = 1 + log\left(\frac{1+|D|}{1+|d_t|}\right) \quad (2)$$

where $|d_t|$ is the number of documents containing the term t, i.e. the number of titles or descriptions that include the relevant term. The inverse document frequency is used to penalize very common terms in the corpus (e.g. “issue” or “component”), as they may act as noise to our model.

Task Meta-data Model. Concerning the meta-data model, which for the proof of concept proposed by this paper comprises the type and the assignee of each task, we could immediately apply a classification algorithm. However, by doing so, the temporal characteristics of the data would not be considered. Instead, we implemented a sliding window technique so that the importance of a new task is predicted using the recent tasks, which are intuitively expected to have similar characteristics. Consider, for instance, that a task has an assignee A that lately delivers mostly Critical tasks; assigning him/her a new task at this time would probably mean that the task is Critical and not Minor. Obviously, this is related to the number of open tasks and the number of developers available. However, practice says that when one performs well, then he/she is assigned with relevant tasks, both in context, as well as importance.

Hence, to model the temporal characteristics of numerical values, we use the sliding technique shown in Figure 3. Given that the tasks are ordered according to the date they were created, we initially choose a sliding window size *s* (in our case set to 50 after experimentation). Then, for each new task we use the data from the previous *s* tasks to predict its importance value. For the $s+1$-th task, we use data from the tasks 1 to *s*, for the $s+2$-th task, we use data from the tasks 2 to $s+1$, and so on till the last (*n*-th) task.

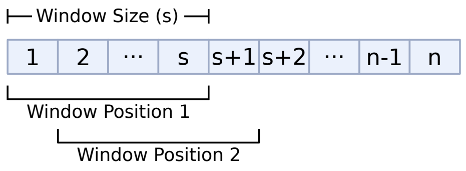


Figure 3: Sliding Window Technique.

3.3 Importance Prediction

Upon having processed our data inputs, we proceed to applying prediction techniques in order to predict the importance of a given task. As there are three inputs, one for titles, one for descriptions, and one for meta-data, we build three models that we later aggregate to produce a single importance value (as shown in Figure 1). We apply simple averaging for the aggregation since it is adequate as a proof of concept that using more information leads to better results.

An important consideration for selecting a prediction technique is the fact that the output has ordinal characteristics. In other words, the relative ordering between different importance classes is significant. Thus, we expect that using an algorithm that considers the order of the classes can be more effective that using a typical classification algorithm. To test this hypothesis, we have applied three algorithms, all based on Support Vector Machines (SVM), which are described in the following subsections.

Classification. Our first model is the SVM classifier provided by scikit-learn (Pedregosa et al., 2011) with an RBF kernel and default parameter values, i.e. the Support Vector Classifier (SVC). The algorithm follows the one-versus-one approach to multiclass classification. According to that approach, given *c* classes, we build $(c\cdot(c-1))/2$ binary classifiers, one for each pair of classes. For the classes Trivial, Minor, Critical, and Blocker, we have six classifiers: *Trivial-vs-Minor*, *Trivial-vs-Critical*, *Trivial-vs-Blocker*, *Minor-vs-Critical*, *Minor-vs-Blocker*. After training and executing the models, the resulting class is determined using majority voting.

Regression. A straightforward way to model ordinal output variables is by using regression techniques, i.e. Support Vector Regression (SVR). In regression, the dependent variable takes continuous values, which obviously have ordering. In our case, the SVR model outputs values in the range $[1,4]$. The next step is to translate the continuous output values into class probabilities. To do so, we use the following equation:

$$y_i = \frac{\frac{1}{i-y_r}}{\frac{1}{1-y_r}+\frac{1}{2-y_r}+\frac{1}{3-y_r}+\frac{1}{4-y_r}} \tag{3}$$

where y_r is the output of the SVR, and y_i is the final probability of class i. Finally, the class predicted by the algorithm is the one with the maximum probability, i.e. $argmax_i(y_i)$.

Ordinal Classification. For our third classifier, we employed the ordinal classification approach proposed in (Frank and Hall, 2001). The technique includes one less model than the class variables, with each of them predicting the probability of being larger than one of the first three class values. In our case, the three models provide as output the values of $P(y > Trivial)$, $P(y > Minor)$, and $P(y > Critical)$. After that, the probability of each class is provided by the following set of equations:

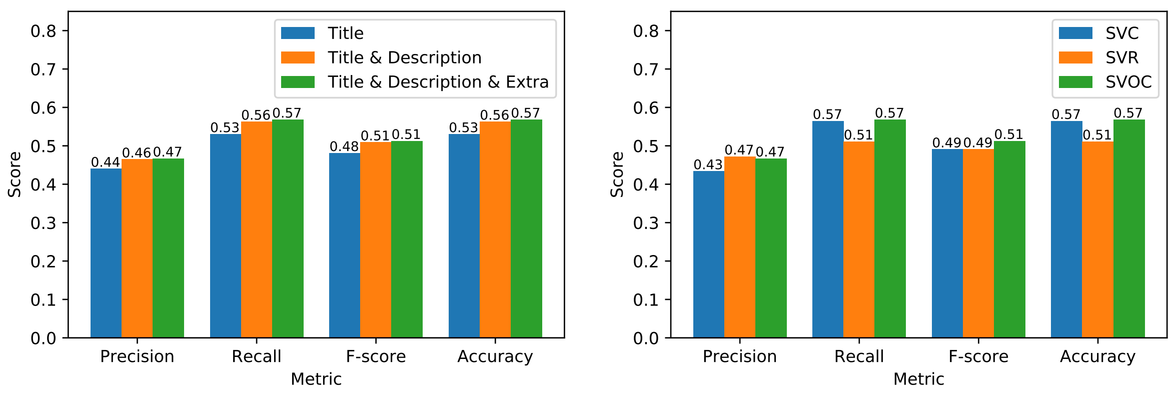
$$\begin{aligned}
&P(y = Trivial) = 1 - P(y > Trivial)\\
&P(y = Minor) = P(y > Trivial) - P(y > Minor)\\
&P(y = Critical) = P(y > Minor) - P(y > Critical)\\
&P(y = Blocker) = P(y > Critical)
\end{aligned} \tag{4}$$

Finally, concerning the output of this Support Vector Ordinal Classifier (SVOC), the class predicted by the algorithm is the one with the maximum probability, i.e. $argmax_c(P(y=c))$.

4 EVALUATION

4.1 Evaluation Framework

To illustrate the applicability of our claims and provide an initial proof of concept for them, we have selected a dataset of 10 projects. These projects were selected as they are large enough to provide useful data (e.g. all have more than 2000 tasks) and have multiple importance values without excessive class imbalances. The projects of our evaluation dataset, along with certain statistics about their tasks and contributors are shown in Table 2. In all projects, we use the first 80% of the data as training set and the most recent 20% of the data as test set.



(a) (b)

Figure 4: Evaluation Results, including (a) Comparison of Data Models, and (b) Comparison of Importance Classifiers.

Table 2: Evaluation Dataset of Jira Projects.

Project	#Tasks	#Contributors
AS7	5428	676
CLOUDSTACK	5494	399
COUCHDB	2004	577
GRAILSPLUGINS	2923	985
ISPN	3836	314
JBESB	3870	251
JBPAPP	5759	398
MAPREDUCE	5478	751
WFLY	2469	464
XERCESC	2022	1238

In order to evaluate our hypothesis, we have performed two experiments. The first aspires to evaluate the data models described in subsection 3.2 while the second attempts to assess the effectiveness of the three importance prediction techniques of subsection 3.3. For both experiments, we used the classification metrics of Precision, Recall, F-score, and Accuracy.

4.2 Evaluation of Data Models

Concerning the evaluation of the different data models, we keep only SVOC and test three configurations, one using only the titles of the tasks, one using the titles and the descriptions, and one using all available data (i.e. including also the meta-data). The results of our analysis are shown in Figure 4a. Since tasks are split into four classes of importance, our results seem effective enough for a task importance recommender. Moreover, given this graph, we confirm our intuition that employing more data leads to better results.

It is clear that using also the descriptions of the tasks instead of only the titles results in higher values concerning all four evaluation metrics. The results also indicate that meta-data can offer useful information; the data model that uses all data seems to perform more effectively than the other two models in all metrics. This outcome is rather expected, considering that the type and the assignee of the task are indicative of its importance. Finally, given these promising results, as future work, one may consider extending our model to the rest of the data provided for each task.

4.3 Evaluation of Importance Classifiers

The results of the evaluation of our three importance classifiers are shown in Figure 4b. At first glance, one may notice that the regression algorithm (SVR) does not perform as effectively as the other two techniques. SVR seems to achieve high precision (and, thus, f-score), however its values on recall and accuracy are clearly lower than those of the other two algorithms. This is actually expected as regression algorithms are generally not suitable for classification problems. Although in our case importance is an ordinal variable, it is certainly not continuous.

Concerning the other two implementations, our ordinal classifier (SVOC) seems to outperform the SVC technique when it comes to precision (and, therefore, F-score). For the recall and accuracy metrics, the two algorithms have similar effectiveness. This is actually quite an interesting outcome, indicating that SVOC successfully models the output class, and thus can be used effectively for ordinal classification problems such as the one analyzed in this work.

5 CHALLENGES IN TASK IMPORTANCE PREDICTION

Upon laying the foundation of the task importance prediction challenge and providing an initial proof of concept, we discuss any limitations, along with open issues to be addressed. First and foremost, we note that research works for the challenge itself are quite limited. As already mentioned in Section 2, most research efforts focus on bug severity prediction, a challenge that may be similar, however concerns mainly the maintenance phase of the software development process. For the same reason, project/task management datasets are also few, especially compared to their vast counterparts originating from bug tracking systems (Lamkanfi et al., 2013). An interesting idea in this context would be to attempt to extract issues from code hosting services, such as GitHub, which are also often used only for bugs, however they are also sometimes used to track feature development (Diamantopoulos et al., 2020).

Another important note supported by our evaluation results is the fact that employing both textual information and meta-data can lead to better importance predictions. This observation is on par with current research in bug severity prediction (Tian et al., 2012), while it also conforms with approaches in relevant research fields, such as automated issue/bug assignment (Matsoukas et al., 2020; Anvik et al., 2006). Moreover, our intuition tells us that using temporal characteristics can further improve on the results; despite not proven explicitly in this paper, following an holistic approach should provide more effective predictions if we consider similar works (Tian et al., 2015).

Concerning the output, in this work we have considered projects that did not exhibit excessive imbalance among the task importance classes. We have, of course, removed any default instances, as supported by current research (Menzies and Marcus, 2008), however we would certainly propose a more rigorous analysis for other projects. We have focused mainly on the ordinal characteristics of the output and tested three different importance prediction scenarios. Our evaluation results indicate an ordinal classification method to be the most efficient, whereas a typical SVM classifier also had acceptable results. On the other hand, using regression does not seem to be a good fit for this problem.

Concerning our methodology, we note that it provides interesting paths for future research. As our main purpose has been to provide a proof of concept, we have made certain assumptions as to the data and the model parameters. In specific, we selected open-source projects with multiple contributors, as these are representative of contemporary practice in software development. We have selected projects that do not exhibit class imbalance, and further assumed that task types with few occurrences are outliers and thus dropped the relevant tasks. Moreover, the size of the sliding window for the temporal characteristics was set arbitrarily upon experimentation; an interesting extension would be to set this value in a temporal manner (e.g. considering the tasks of the past month) or even according to the specifics/status of each project (e.g. maintenance periods have been shown to exhibit different characteristics from active development periods (Papamichail et al., 2019)). Finally, there is of course ample room for further research concerning the types of models involved as well as the way they are combined; e.g. one could employ a single model for all input variables (both textual and meta-data) or try using text embeddings and/or neural networks.

6 CONCLUSIONS

As software development is more and more becoming a collaborative process that is supported by project/task management systems, the need for automation is more imminent than ever. In this paper, we have described the challenge of task importance prediction, of which the automation can certainly save valuable time and effort during the software development process. As a proof of concept, our methodology has been proven effective for task importance prediction and has provided interesting points of discussion. Apart from illustrating the benefits of considering both textual and non-textual data, our results have moreover shown that the ordinal characteristics of the importance variable should be taken into account.

As future work, we plan to improve on all aspects of our methodology, while focusing on the challenges described in Section 5. In specific, concerning our data model, one could employ more information including not only the type and the assignee of each task, but also its labels, components, etc. Moreover, we could investigate the effectiveness of different modeling techniques (including e.g. text embeddings instead of the tf-idf vector space model) as well as different aggregation techniques (instead of averaging over the results of the classifiers). Other areas of focus include addressing the class imbalance that may be found in different projects as well as further experimenting on the importance prediction algorithms (e.g. by also evaluating an one-vs-many SVC or even testing other types of algorithms).

ACKNOWLEDGEMENTS

This work has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code: T1EDK-04045).

REFERENCES

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who Should Fix This Bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA. ACM.

Cohen, W. W. and Singer, Y. (1999). A Simple, Fast, and Effective Rule Learner. In *Proceedings of the 16th National Conference on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 335–342, USA. American Association for Artificial Intelligence.

Diamantopoulos, T., Papamichail, M., Karanikiotis, T., Chatzidimitriou, K., and Symeonidis, A. (2020). 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, pages 558–562, Seoul, South Korea. ACM.

Frank, E. and Hall, M. (2001). A Simple Approach to Ordinal Classification. In *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, pages 145–156, Berlin, Heidelberg. Springer-Verlag.

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

Lamkanfi, A., Demeyer, S., Giger, E., and Goethals, B. (2010). Predicting the Severity of a Reported Bug. In *2010 7th IEEE Working Conference on Mining Software Repositories*, MSR '10, pages 1–10. IEEE Press.

Lamkanfi, A., Demeyer, S., Soetens, Q. D., and Verdonck, T. (2011). Comparing Mining Algorithms for Predicting the Severity of a Reported Bug. In *Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering*, CSMR '11, pages 249–258, USA. IEEE Computer Society.

Lamkanfi, A., Pérez, J., and Demeyer, S. (2013). The Eclipse and Mozilla Defect Tracking Dataset: A Genuine Dataset for Mining Bug Information. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 203–206. IEEE Press.

Matsoukas, V., Diamantopoulos, T., Papamichail, M., and Symeonidis, A. (2020). Towards Analyzing Contributions from Software Repositories to Optimize Issue Assignment. In *2020 IEEE International Conference on Software Quality, Reliability and Security*, QRS 2020, pages 243–253, Vilnius, Lithuania. IEEE Press.

Menzies, T. and Marcus, A. (2008). Automated Severity Assessment of Software Defect Reports. In *2008 IEEE International Conference on Software Maintenance*, ICSM 2008, pages 346–355. IEEE Press.

Ortu, M., Destefanis, G., Adams, B., Murgia, A., Marchesi, M., and Tonelli, R. (2015). The JIRA Repository Dataset: Understanding Social Aspects of Software Development. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE '15, New York, NY, USA. ACM.

Papamichail, M. D., Diamantopoulos, T., Matsoukas, V., Athanasiadis, C., and Symeonidis, A. L. (2019). Towards Extracting the Role and Behavior of Contributors in Open-source Projects. In *Proceedings of the 14th International Conference on Software Technologies*, ICSOFT 2019, pages 536–543, Prague, Czech Republic. SciTePress.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Roy, N. K. S. and Rossi, B. (2014). Towards an Improvement of Bug Severity Classification. In *Proceedings of the 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, SEAA '14, pages 269–276, USA. IEEE Computer Society.

Sharma, M., Bedi, P., Chaturvedi, K. K., and Singh, V. B. (2012). 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, pages 539–545. IEEE Press.

Tian, Y., Lo, D., and Sun, C. (2012). Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, WCRE '12, pages 215–224, USA. IEEE Computer Society.

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

Uddin, J., Ghazali, R., Deris, M. M., Naseem, R., and Shah, H. (2017). A Survey on Bug Prioritization. *Artif. Intell. Rev.*, 47(2):145–180.

Yang, C.-Z., Hou, C.-C., Kao, W.-C., and Chen, I.-X. (2012). 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, pages 240–249, USA. IEEE Computer Society.

Yang, G., Zhang, T., and Lee, B. (2014). 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, pages 97–106, USA. IEEE Computer Society.