Towards Automated Prediction of Software Bugs from Textual Description

Suyash Shukla and Sandeep Kumar
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India

Keywords: Issue Tracking System, Machine Learning, Term Frequency Inverse Document Frequency, Smartshark.

Abstract: Every software deals with issues such as bugs, defect tracking, task management, development issue to a customer query, etc., in its entire lifecycle. An issue-tracking system (ITS) tracks issues and manages software development tasks. However, it has been noted that the inferred issue types often mismatch with the issue title and description. Recent studies showed machine learning (ML) based issue type prediction as a promising direction, mitigating manual issue type assignment problems. This work proposes an ensemble method for issue-type prediction using different ML classifiers. The effectiveness of the proposed model is evaluated over the 40302 manually validated issues of thirty-eight java projects from the SmartSHARK data repository, which has not been done earlier. The textual description of an issue is used as input to the classification model for predicting the type of issue. We employed the term frequency-inverse document frequency (TF-IDF) method to convert textual descriptions of issues into numerical features. We have compared the proposed approach with other widely used ensemble approaches and found that the proposed approach outperforms the other ensemble approaches with an accuracy of 81.41%. Further, we have compared the proposed approach with existing issue-type prediction models in the literature. The results show that the proposed approach performed better than existing models in the literature.

1 INTRODUCTION

The maintenance of any software system is crucial as it involves activities such as mitigating potential defects in the source code, the evolvement of software based on user requirements, etc. Issue tracking systems such as JIRA, Github, etc., are tools that support these maintenance tasks by efficiently managing and controlling issues arising in the software. The software developers or users label the issue in the system, which helps maintain the software (Alonso-Abad et al., 2019). However, it is noticeable from the existing research that reported issue types usually differ in their title, and description (Antoniol et al., 2008; Herzig et al., 2013a). Misclassified issues can negatively affect the software development process and users who use the issue-tracking system data. Researchers conducted experiments over different datasets (Herzig et al., 2013a; Herbold et al., 2020a) and reported that almost 40% of the issues are misclassified as bugs.

One way to deal with the problem of mislabelling is by using machine learning (ML) techniques to predict issue types from the title and description of the issue. In the past, the researchers used different supervised(Antoniol et al., 2008; Chawla and Singh, 2015; Zhou et al., 2016; Otoom et al., 2019; Kallis et al., 2019) and unsupervised (Hammad et al., 2018; Chawla and Singh, 2018) learning models for automated issue-type prediction over different datasets. The classifiers provide good prediction accuracy; however, there are chances that the predicted issue types might still be misclassified. This problem can be alleviated by training the model with manually verified issue data. To handle these gaps, we have presented a methodology for the prediction of issue type from a textual description of the issue. Further, this work proposes an ensemble approach for issue type prediction over the 40302 manually validated issues of thirty-eight java projects from the SmartSHARK data repository. The proposed ensemble model uses naive Bayes (NB), K-nearest neighbor (KNN), decision tree (DT), and logistic regression (LR) as base estimators and support vector classifier (SVC) as the meta estimator. The textual descriptions of various issues are used as input to the proposed model for predicting its type. The textual data is converted into numerical features using the term frequency-inverse

Shukla, S. and Kumar, S.
Towards Automated Prediction of Software Bugs from Textual Description.
DOI: 10.5220/0011982400003464
In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2023)*, pages 193-201
ISBN: 978-989-758-647-7; ISSN: 2184-4895

document frequency (TF-IDF) method and provided to the proposed model.

The following are some of the contributions made by the work presented:

- We extracted the data of different java projects from the SmartSHARK release 2.2 data repository and preprocessed them. Hence, through this work, we have contributed a ready-to-use dataset with issues verified by the researchers.
- We have proposed an ensemble-based approach for issue-type prediction. The proposed approach used the TF-IDF method to convert the textual description of different issues into numerical features. These numerical features will serve as input for the issue type prediction.
- We have compared the effectiveness of the proposed approach against different solo and ensemble classification models for issue-type prediction.
- We have also compared the proposed approach to existing models in the literature.

The following research questions will be investigated in the experimental study presented:

RQ1: How does combining issue title and description affects model performance?

RQ2: How effective is the proposed method compared to the base learners utilized for issue-type prediction?

RQ3: How effective is the proposed method compared to the ensemble learners utilized for issue-type prediction?

The remainder article is organized as follows: Section 2 discusses related work for issue-type prediction. The process for data preparation is discussed in Section 3. The overview of the proposed methodology is discussed in section 4. Section 5 discusses the implementation details and results of the experimental analysis. Section 6 discusses the comparative analysis. The answers to the RQs are discussed in section 7. In Section 8, threats to validity are examined. Finally, Section 9 discusses the conclusion.

2 RELATED WORK

Both supervised and unsupervised methods have been used for issue-type prediction. Antoniol et al. (Antoniol et al., 2008) used three classification algorithms (NB, DT, and LR) and the term frequency matric (TFM) feature representation method for issue type prediction. They also used symmetrical uncertainty feature selection to remove irrelevant features. Chawla et al. (Chawla and Singh, 2015) utilized a fuzzy classifier with TFM over the issue title to predict the issue type. Otoom et al. (Otoom et al., 2019) modified the TFM method by introducing a set of 15 words and calculated the term frequencies of those words from the issue title and descriptions. Zhou et al. (Zhou et al., 2016) used the structural information of issues with their title and descriptions for issue type prediction. They also classified titles into three categories (high, low, medium) based on the difficulty of deciding whether it's a bug or not and used them to train the NB classifier. Herbold et al. (Herbold et al., 2020b) developed a methodology incorporating manually specified knowledge for issue-type prediction using ML models. They used ML classifiers to predict issue types over the SmartSHARK dataset. Li et al. (Li et al., 2022) also developed a method for issue type prediction incorporating Long Short-Term Memory as a feature extraction method over the SmartSHARK dataset.

Some researchers used unsupervised learning algorithms by categorizing issues into different clusters to predict issue types. Hammad et al. (Hammad et al., 2018) used an agglomerative hierarchical clustering approach to categorize issues into distinct clusters based on similarity. Then, they identified features for each cluster and built an ML model for each cluster to predict issue types. Chawla et al. (Chawla and Singh, 2018) used fuzzy C-means clustering for the same task.

Based on the above discussion, we can say that both supervised and unsupervised learning models were used in the literature and provided good prediction accuracy. However, there are still misclassified issues, which may create problems for the high-quality software applications which are largely dependent on the information of these issue-tracking systems. This problem requires the manual intervention of researchers to verify developer-assigned issue types. The SmartSHARK data repository used in this work has projects containing manually validated issues by the researchers. The learning model's performance can be improved by evaluating them over manually validated issues. So, this work proposes an ensemble classification model for issue type prediction over manually validated issues from the SmartSHARK data repository.

3 DATASET GENERATION

This work uses the SmartSHARK release 2.2 (Trautsch et al., 2021) dataset for experimentation, which is recently published and used by several researchers (Khoshnoud et al., 2022; de Almeida et al.,

Table 1: Detailed Information on the Dataset used.

Projects	# of Issues	# of Bugs	# of Non-Bugs
Ant_Ivy	1526	912	614
Archiva	1630	1035	595
Calcite	2281	1689	592
Cayenne	2366	1196	1170
Commons_Bcel	305	237	68
Commons_Beanutils	509	322	187
Commons_Codec	240	129	111
Commons_Collections	639	337	302
Commons_Compress	453	263	190
Commons_Configuration	719	432	287
Commons_Dbcp	530	365	165
Commons_Digester	188	118	70
Commons_Io	565	272	293
Commons_Jcs	159	117	42
Commons_Jexl	270	126	144
Commons_Lang	1342	619	723
Commons_Math	1395	667	728
Commons_Net	646	456	190
Commons_Scxml	269	106	163
Commons_Validator	444	257	187
Commons_Vfs	669	417	252
Deltaspike	1034	406	628
Eagle	997	384	613
Giraph	1129	496	633
Gora	535	184	351
Jspwiki	942	554	388
Knox	1383	821	562
Kylin	2810	1511	1299
Lens	1096	490	606
Mahout	1825	755	1070
Manifoldcf	1534	802	732
Nutch	2508	1164	1344
Opennlp	1068	312	756
Parquet_Mr	1138	581	557
Santuario_Java	495	379	116
. Systemml	1476	496	980
Tika	2572	1299	1273
Wss4j	615	329	286
	40302	21035	19267

2022; Peruma et al., 2022). The SmartSHARK dataset contains issue-tracking data, bug-inducing data, mailing lists, pull request data, etc., of 96 java projects, available in larger (640 gigabytes) and smaller forms (65 gigabytes). However, we have only extracted data from thirty-eight java projects from the SmartSHARK release 2.2 data repository because they contain manually validated issues. The highlights of thirty-eight java projects used for this work are shown in Table 1.

To acquire issue data from these projects, first, we downloaded the .agz file of the SmartSHARK dataset, as shown in Figure 1. Then, we created a MongoDB instance and loaded the dataset locally using mongorestore. Finally, we selected the issue tracker of the project under consideration and extracted its issue data. The issue dataset used in this work contains 40302 issues. Out of 40302 issues, 21035 are bugs, and 19267 are nonbugs.

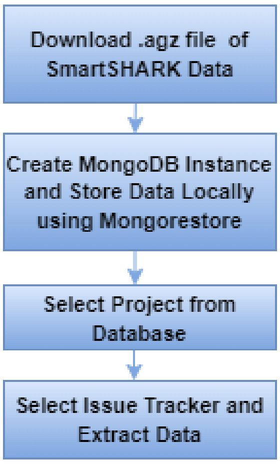

Figure 1: Extraction of Issues from the SmartSHARK Data.

4 PROPOSED METHODOLOGY

The methodology used for issue-type prediction is divided into three steps, as discussed in Figure 2.

4.1 Load Issue Data

This step involves loading the extracted issue data, which contains the issue id, title, description, issue type, status, linked issues, priority, pull request, etc. However, we used only textual data as independent features to predict the issue type for this work. The issue type represents the type of issue that can be of any type, such as bug, task, enhancement, improvement, new feature, etc. But we have considered the issue types other than bugs as non-bugs because this work aims to investigate the classification of bug and non-bug issues. Ready-to-use datasets of our experimental analysis can be found in the GitHub repository. https://anonymous.4open.science/r/ENASE_Research-1BD8.

4.2 Data Preprocessing

The data extracted from the issue tracker is in raw format. So, preprocessing is required to convert data into a usable format. Purposefully, we removed the issues containing missing values for independent or dependent features and followed the below steps:

- **Tokenization of textual descriptions:** This step breaks the textual descriptions into tokens. Then, it removes the unnecessary punctuation from them.
- **Conversion into Lowercase:** This step converts the words (or tokens) in the text to lowercase as the programming languages are case-sensitive and consider 'was' and 'WAS' as different words.
- **Removal of Stopwords:** There are so many words in the natural language that do not represent any useful information when used alone. These words are called stopwords and can be removed from the feature space.
- **Conversion of words into stems:** This is one of the most important steps in topic modeling, which converts the words (or tokens) into their stems. The benefit of stemming is that the words such as stop, stopped, and stopping are considered the same.
- **Convert textual data to numerical data:** After stemming, we used the TF-IDF method to convert the textual data into numerical features to make it suitable for the machine learning model. The TF-IDF is an important technique in information retrieval, which computes the score for each word in the text and signifies its importance.

4.3 Model Training and Testing

After obtaining the numerical features from the issue's textual description, the proposed ensemble model is built over them to predict the type of issue. The proposed model is based on the idea of stacking ensemble, which uses four classification models, such as NB, KNN, DT, and LR as base estimators and SVC as the meta estimator. First, the issue data is given to the base estimators to generate intermediate predictions, which will then be fed to the meta-estimator to generate final predictions. These final predictions will be used to evaluate the performance of the proposed model.

5 EXPERIMENTAL ANALYSIS

5.1 Implementation Details

This subsection discusses the implementation details used for the experimental analysis. For experimentation, we have chosen the most commonly used classifiers in this domain, i.e., NB, KNN, DT, LR, and SVC. NB and KNN are selected due to their simplicity and easy-to-implement nature, DT and LR are chosen due to their versatility, and SVC is chosen as it minimizes the risk of overfitting. To evaluate the performance of the proposed model, we divided the issue data into the ratio 70:30, 70% of the data is used for model training, whereas 30% of the data is used for model testing. We have chosen the default hyperparameters

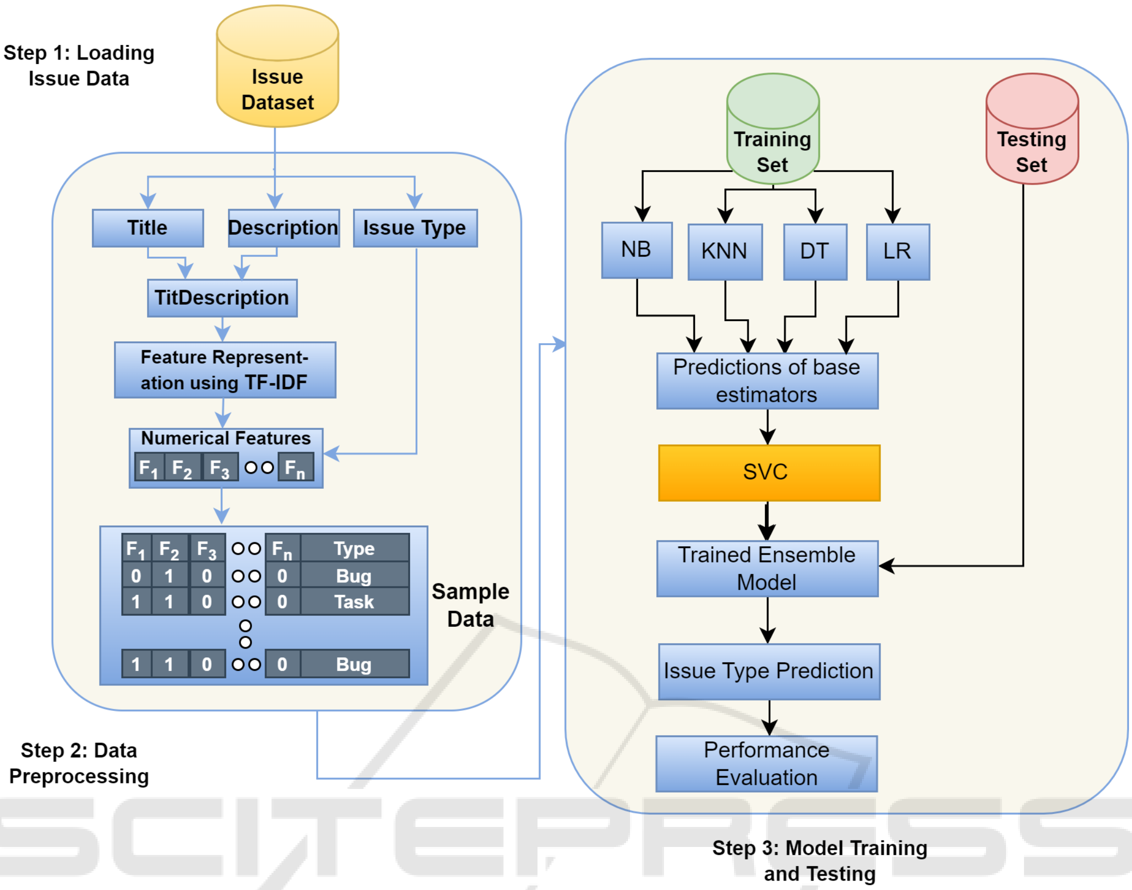

Figure 2: Extraction of Issues from the SmartSHARK Data.

Table 2: Details of performance measures utilized.

Measure	Description	Formula
Accuracy (A)	It is the proportion of correctly predicted issues compared to all issues.	$A = \frac{Correctly\ Predicted\ Issues}{All\ Issues}$
Precision (P)	It is the proportion of correctly predicted bugs compared to all bugs.	$P = \frac{Correctly\ Predicted\ Bugs}{All\ Bugs}$
Recall (R)	It is the proportion of correctly predicted bugs compared to all predicted bugs.	$R = \frac{Correctly\ Predicted\ Bugs}{All\ Predicted\ Bugs}$
F1-score (F1)	It is defined as the harmonic mean of precision and recall.	$F1 = 2\frac{P*R}{P+R}$

for each classification model. The implementation of different ML techniques is done using the scikit-learn library of python.

5.2 Performance Measures

The earlier works on issue-type prediction used precision, recall, and f1-score to evaluate classifier performance (Antoniol et al., 2008; Kallis et al., 2019; Herzig et al., 2013b; Hosseini et al., 2017; Just et al., 2014; Lukins et al., 2008; Marcus et al., 2004). We have also used these performance measures in this work. Apart from these three measures, we have also used the accuracy (Mills et al., 2018) measure to evaluate different classifiers. The formula and description of these measures are shown in Table 2.

5.3 Experimental Results

This subsection presents the experimental results of this work on applying the proposed ensemble approach over the 40302 manually validated issues of thirty-eight java projects from the SmartSHARK data repository. The performance of the proposed ensem-

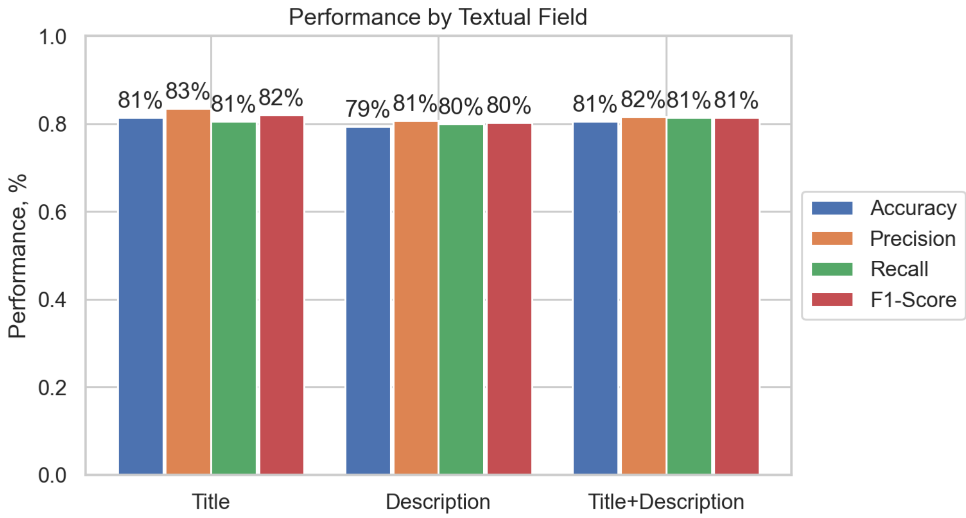

Figure 3: Performance of the proposed model over different textual fields.

ble model is evaluated under three scenarios, (i) Only title as input to the proposed model, (ii) Only description as input to the proposed model, (iii) Combination of title and description as input to the proposed model. The performance of the proposed model for manually validated issues is shown in Figure 3.

From Figure 3, we can say that the proposed model performed well when only considering the issue title as the input to the model with an accuracy of 81.41%, demonstrating that the issue title contains more useful information to predict its type. Also, the concatenation of the issue title and description is not effectively using the information of two fields as the performance is decreasing. Table 3 shows the confusion matrix obtained after applying testing data to the proposed model.

Table 3: Confusion matrix for the testing data.

	Bug	**NonBug**
Bug	5111	1231
NonBug	1016	4733

6 COMPARATIVE ANALYSIS

6.1 Comparison with Other Models

This subsection compares the performance of the proposed model with the base learners (NB, KNN, DT, and LR) and widely used ensemble learners such as the bagging classifier, AdaBoost classifier, random forest classifier (RFC), gradient boosting classifier (GBC), and extra tree classifier. The comparison of the proposed model against the other models is shown in Table 4.

From Table 4, we can say that the proposed model outperformed the base learners used in this work. The accuracy of the base learners lies in 0.6605-0.8035, whereas the accuracy of the proposed model is 0.8141. Similarly, the accuracy of the ensemble learners lies in 0.7409-0.8113, which is lower than the proposed model. We have used Wilcoxon's signed rank (Seo and Bae, 2013) test for the statistical analysis as it does not require the data to follow any distribution. The Wilcoxon test compares the relative performance of two models depending on the p-value; a p-value less than 0.05 shows the models are significantly different, whereas a p-value greater than 0.05 indicates no difference among the models. Wilcoxon test results for different classifiers used in this work are shown in Table 5.

Table 5 shows that the proposed model differs significantly from the other models, as the p-values are less than 0.05.

6.2 Comparison with Existing Works

This section compares the performance of the proposed approach with the existing works (Otoom et al., 2019; Kallis et al., 2019; Herbold et al., 2020b; Pandey et al., 2018; Limsettho et al., 2014) on issue-type prediction in the literature. The existing works mentioned above have used datasets other than the one used in this work, as the SmartSHARK (current release) data-based works are not available in the literature. Limsettho et al.(Limsettho et al., 2014), Otoom et al. (Otoom et al., 2019), and Pandey et al. (Pandey et al., 2018) have used three OSS project

Table 4: Comparison of the proposed model with the base learners.

	Technique	A	P	R	F1
Base Learners	NB	0.782648251	0.777495517	**0.820403658**	0.798373485
	KNN	0.660573981	0.766555503	0.507410911	0.610626186
	DT	0.738565875	0.759419344	0.734153264	0.746572597
	LR	0.803572905	0.831301152	0.784768212	0.80736475
Ensemble Learners	Bagging	0.774046812	0.784790155	0.784295175	0.784542587
	AdaBoost	0.740964354	0.792136877	0.686218858	0.735383576
	RFC	0.794144405	0.811278074	0.791706086	0.801372596
	GBC	0.753535688	0.800071403	0.706717124	0.750502344
	Extra tree	0.811347283	0.826499437	0.810469883	0.818406178
Proposed Model		**0.814159**	**0.834177**	0.805897	**0.819793**

Table 5: P-values for different models.

	P-value	H-stat	Significance
Proposed vs NB	8.201e-05	10.0	Yes
Proposed vs KNN	1.907e-06	0.0	Yes
Proposed vs DT	1.907e-06	0.0	Yes
Proposed vs LR	0.0239	45.0	Yes
Proposed vs Bagging	1.907e-06	0.0	Yes
Proposed vs AdaBoost	1.907e-06	0.0	Yes
Proposed vs RFC	0.00315	29.0	Yes
Proposed vs GBC	1.907e-06	0.0	Yes
Proposed vs Extra tree	0.03123	41.5	Yes

datasets from the Apache repository. Kalis et al. (Kallis et al., 2019) used a dataset of 30000 issues from Github, while Herbold et al. (Herbold et al., 2020b) used the earlier release of the SmarkSHARK dataset. The F1-score for the existing works lies in 0.65-0.805, whereas the F1-score of the proposed method is 0.819793. So, we can say that the results obtained from the proposed model are significantly improved compared to the existing works in the literature.

7 DISCUSSION

This section discusses answers to the research questions.

RQ1: How does combining issue title and description affects model performance?

To answer this RQ, we evaluated the performance of the proposed ensemble model is evaluated under three scenarios, (i) title as input to the proposed model, (ii) description as input to the proposed model, (iii) a combination of title and description as input to the proposed model, shown in Figure 3. Figure 3 shows that the proposed model performed well when only considering the issue title as the input to the model with an accuracy of 81.41%, demonstrating that the issue title contains more useful information to predict its type.

RQ2: How effective is the proposed method compared to the base learners utilized for issue-type prediction?

To answer this RQ, we compared our model's performance to that of the basic learners (NB, KNN, DT, and LR) employed in this study, as shown in Table 4. The suggested model performed better than the basic learners utilized in this work, as shown in Table 4. The proposed model's accuracy is 0.8141 as opposed to the base learners' accuracy, which ranges from 0.6605-0.8035.

RQ3: How effective is the proposed method compared to the ensemble learners utilized for issue-type prediction?

To answer this RQ, we have compared the performance of the proposed model with the widely used ensemble learners such as the bagging classifier, AdaBoost classifier, RFC, GBC, and extra tree classifier, as shown in Table 4. Table 4 shows that the proposed model outperformed the ensemble learners used in this work. The accuracy of the ensemble learners lies in 0.7409-0.8113, whereas the accuracy of the proposed model is 0.8141.

8 THREATS TO VALIDITY

This section discusses threats related to validity.
Internal Validity: Internal validity is primarily jeopardised by the possibility of errors in the implementation of the proposed and compared approaches in the study. To reduce this risk, we build the proposed and compared approaches on mature frameworks/libraries (such as Jupyter and sklearn) and thoroughly test our code and experiment scripts before and during the experimental study. Another risk may be posed by parameter settings for the investigated methods. However, for the learning models investigated in this paper, we used default hyperparameters.
External Validity: External validity discusses the generalizability of the results of this work. This work uses only thirty-eight java projects, and their data is gathered from the JIRA issue tracker. So, the results of this work may not apply to projects developed in different programming languages whose data is collected from other issue trackers.
Construct Validity: Construct validity discusses the performance measures used in this work. The presented work uses four performance measures: accuracy, precision, recall, and F1-score, for model evaluation by ignoring the other important measures, which may affect the results of this work. Although we have verified with different works in the literature and found that the measures used in this work are the most popular measures for issue type prediction.

9 CONCLUSION

The issue-tracking systems are useful for software maintenance activity. However, the incorrect classification of issues may create problems for high-quality software applications. This problem requires the manual intervention of researchers to verify developer-assigned issue types. An ample amount of research has been done for issue-type prediction over different datasets in the past. However, the used datasets do not have manually verified issues. In this work, we used the SmartSHARK release 2.2 dataset containing manually verified projects for analysis and handled various challenges related to the dataset mentioned above. Further, we proposed an ensemble-based approach and evaluated its performance over the 40302 manually validated issues of thirty-eight java projects from the SmartSHARK data repository. The results show that the proposed model performed well when considering only the issue title as the input. Further, we have compared the proposed approach with other models and found that the proposed approach showed significant improvement compared to the other models.

REFERENCES

Alonso-Abad, J., López-Nozal, C., Maudes-Raedo, J., and Marticorena-Sánchez, R. (2019). Label prediction on issue tracking systems using text mining. *Progress in Artificial Intelligence*, 8(3):325–342.

Antoniol, G., Ayari, K., Penta, M. D., Khomh, F., and Gueheneuc, Y. (2008). Is it a bug or an enhancement?: A text-based approach to classify change requests. In *In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, pages 304–318.

Chawla, I. and Singh, S. (2015). An automated approach for bug categorization using fuzzy logic. In *In: Proceedings of the 8th India Software Engineering Conference*, pages 90–99.

Chawla, I. and Singh, S. (2018). Automated labeling of issue reports using semisupervised approach. *Journal of Computational Methods in Sciences and Engineering*, 18(1):177–191.

de Almeida, C., Feijó, D., and Rocha, L. (2022). Studying the impact of continuous delivery adoption on bug-fixing time in apache's open-source projects. In *In Proceedings of the 19th International Conference on Mining Software Repositories*, pages 132–136.

Hammad, M., Alzyoudi, R., and Otoom, A. (2018). Automatic clustering of bug reports. *International Journal of Advanced Computer Research*, 8(39):313–323.

Herbold, S., Trautsch, A., and Trautsch, F. (2020a). Issues with szz: An empirical assessment of the state of practice of defect prediction data collection. In *arXiv preprint arXiv:1911.08938.*

Herbold, S., Trautsch, A., and Trautsch, F. (2020b). On the feasibility of automated prediction of bug and non-bug issues. *Empirical Software Engineering*, 25(6):5333–5369.

Herzig, K., Just, S., and Zeller, A. (2013a). It's not a bug, it's a feature: How misclassification impacts bug prediction. In *In: Proceedings of the International Conference on Software Engineering*, page 392–401.

Herzig, K., Just, S., and Zeller, A. (2013b). It's not a bug, it's a feature: How misclassification impacts bug prediction. In *In: Proceedings of the International Conference on Software Engineering*, page 392–401.

Hosseini, S., Turhan, B., and Gunarathna, D. (2017). A systematic literature review and meta-analysis on cross-project defect prediction. *IEEE Transactions on Software Engineering*, 45(2):111–147.

Just, R., Jalali, D., and Ernst, M. (2014). Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *In: Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*, pages 437–440.

Kallis, R., Sorbo, A., Canfora, G., and Panichella, S. (2019). Ticket tagger: Machine learning-driven issue classification. In *In: IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 406–419.

Khoshnoud, F., Nasab, A., Toudeji, Z., and Sami, A. (2022). Which bugs are missed in code reviews: An empirical study on smartshark dataset. In *In Proceedings of the 19th International Conference on Mining Software Repositories*, pages 137–141.

Li, Z., Pan, M., Pei, Y., Zhang, T., Wang, L., and Li, X. (2022). Deeplabel: Automated issue classification for issue tracking systems. In *In: 13th Asia-Pacific Symposium on Internetware*, pages 231–241.

Limsettho, N., Hata, H., and Matsumoto, K. (2014). Comparing hierarchical dirichlet process with latent dirichlet allocation in bug report multiclass classification. In *15Th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD)*, pages 1–6.

Lukins, S., Kraft, N., and Etzkorn, L. (2008). Source code retrieval for bug localization using latent dirichlet allocation. In *In: Proceedings of the 2008 15th Working Conference on Reverse Engineering*, pages 155–164.

Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J. (2004). An information retrieval approach to concept location in source code. In *In: Proceedings of the 11th Working Conference on Reverse Engineering*, page 214–223.

Mills, C., Pantiuchina, J., Parra, E., Bavota, G., and Haiduc, S. (2018). Are bug reports enough for text retrieval-based bug localization? In *In: IEEE Int. Conf. on Software Maintenance and Evolution (ICSME)*, page 381–392.

Otoom, A., Al-jdaeh, S., and Hammad, M. (2019). Automated classification of software bug reports. In *In: Proceedings of the 9th International Conference on Information Communication and Management*, pages 17–21.

Pandey, N., Hudait, A., Sanyal, D., and Sen, A. (2018). Automated classification of issue reports from a software issue tracker. In *Progress in intelligent computing techniques: theory, practice, and applications*, page 423–430.

Peruma, A., AlOmar, E., Newman, C., Mkaouer, M., and Ouni, A. (2022). Refactoring debt: Myth or reality? an exploratory study on the relationship between technical debt and refactoring. In *In Proceedings of the 19th International Conference on Mining Software Repositories*, pages 127–131.

Seo, Y. and Bae, D. (2013). On the value of outlier elimination on software effort estimation research. *Empirical Software Engineering*, 18(4):659–698.

Trautsch, A., Trautsch, F., and Herbold, S. (2021). SmartSHARK 2.2 Small.

Zhou, Y., Tong, Y., Gu, R., and Gall, H. (2016). Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, 28(3):150–176.