GCN-Based Issues Classification in Software Repository

Bader Alshemaimri
King Saud University
Riyadh, Saudi Arabia
balshemaimri@ksu.edu.sa

Nafla Alrumayyan
Imam Mohammad Ibn Saud Islamic University
Riyadh, Saudi Arabia
naflaru@gmail.com
nsalrumayyan@imamu.edu.sa

Reem Alqadi
Qassim University
Qassim, Saudi Arabia
re.alqadi@qu.edu.sa

Abstract

Graph Convolutional Network (GCN) have demonstrated significant potential in various fields, particularly in classification tasks. This study introduces GCN-based methodology for classifying issues in software repositories, highlighting advancements in agile software development. Utilizing the dataset by Tawosi et al.(Tawosi et al. 2022), our research demonstrates the potential of GCNs to accurately categorize software issues into bugs, improvements, and tasks. Our results indicate a significant improvement in issue classification, especially for bugs. Additionally, we explore Fast_Text_GCN model, underlining their efficiency in handling dynamic, evolving datasets. This paper contributes to the fields of software engineering and machine learning, offering novel insights into enhancing issue management in software projects.

Introduction

Efficient issue classification is essential during software development to ensure timely problem resolution and effective project management. With the widespread use of Issue Tracking Systems like JIRA and GitHub, software teams are currently facing a situation where they encounter thousands of issues that require precise classification for effective handling. However, the usual method of manual classification of these issues is a labor-intensive and error-prone process, often leading to inefficiencies and delays in project timelines.

While there have been advances in automating various aspects of issue management, a significant gap remains in the automated classification of issues. Addressing this gap, our study introduces an innovative approach utilizing Graph Convolutional Networks (GCNs) for the automatic classification of issues in software systems. GCNs have shown remarkable success in classification tasks across diverse fields (Zhou et al. 2020), indicating their potential utility in enhancing the process of issue classification in software development.

This research aims to explore the efficacy of GCNs in the scope of issue classification within agile software development environments. Leveraging the Tawosi et al. (Tawosi et al. 2022), which includes issues from 39 open-source projects, our study seeks to evaluate the capability of GCNs in this new application domain. The dataset, comprising 458,232 issues from 12 public Jira repositories, offers a comprehensive and varied set of data for analysis. The core objectives of our research are encapsulated in the following research questions:

- **RQ1:** How can Graph Convolutional Networks be employed to enhance the accuracy of issue classification in agile software development projects?
- **RQ2:** How can inductive learning approaches of fast GCNs models improve the performance of GCNs models for issue classification in agile software development projects?

To our knowledge, this is the first investigation into the use of GCNs for issue classification within the framework of the TAWOS dataset, marking a significant step forward in the intersection of software engineering and advanced machine learning techniques.

The structure of this study is organized as follows: Section II presents the Literature Review, where we examine relevant existing research. Section III details our methodology, providing an in-depth explanation of the processes. Section IV is dedicated to the execution of experiments and the subsequent analysis of the results obtained. Finally, Section V concludes the paper, summarizing our findings and outlining directions for future research.

Background

Definition of GCNs

GCNs are a type of neural network designed to work directly with graph-structured data. They are particularly useful in scenarios where data points (nodes) are interconnected or have relationships (edges), like social networks, molecular structures, or recommendation systems.

Transductive vs. Inductive Training

In the context of machine learning, particularly with graph-based methods like GCNs, there are two general approaches to training: transductive and inductive.

Transductive Training This approach typically involves training a model on a specific graph, with the assumption that all nodes of the graph (including those used for testing) are available during training. The model learns the specific structure and node features of this graph, which can limit its ability to generalize to unseen graphs or nodes.(Rossi et al. 2018)

Inductive Training In contrast, inductive training involves training a model in such a way that it can generalize to unseen nodes or entirely new graphs. This is achieved by focusing on learning more generalized node representations based on node features and local structures, rather than relying on the global structure of a specific training graph.(Rossi et al. 2018)

Literature Review

Issue classification in software systems is vital for effective project management. Various studies have explored this, focusing on predicting issue attributes and future actions, such as resolution status or likelihood of reopening. These efforts also delve into classifying issue types and prioritizing urgent issues. Several studies have investigated the use of machine learning models for issue classification. Kallis et al. introduced TicketTagger, a tool employing FastText, a machine learning model, to classify issues into bugs, enhancements, and questions - the most prevalent labels on GitHub(Kallis et al. 2019). This tool achieved high precision and recall, demonstrating the efficacy of machine learning in issue classification. Siddiq and Santos advanced this field further by employing a BERT-based model for issue type prediction, achieving an impressive F1-score (Siddiq and Santos 2022).

Recent research has focused on leveraging deep learning models for issue classification and prioritization. Similarly, Izadi et al. combined machine learning with natural language processing, achieving notable accuracy with a Transformer-based RoBERTa model and a Random Forest classifier (Izadi, Akbari, and Heydarnoori 2022).

Alhindi et al. developed 'Issue-Labeler,' a Jira plugin that automates issue labeling using a deep neural language model (Alhindi et al. 2023), which utilized Google's ALBERT model, showed promising results in classifying issues into bugs, improvements, or new features. Huang et al. adopted a CNN-based approach for 'intention mining' in developer discussions, further showcasing the flexibility of neural networks in understanding issues (Huang et al. 2020).

In recent research conducted on software repositories, a GCNs model, specifically the TextGCN model, was applied to predict the priority of bug reports (Fang et al. 2021). A crucial aspect of this method is the development of a weighted loss function during the training phase, specifically designed to tackle the issue of imbalanced data. The loss function incorporates a label penalty mechanism, aimed at achieving balanced prediction results across various classes. Significantly, this approach has been effective in maintaining high F-measure scores, particularly for classes with a larger volume of samples. In our study, we adapt this approach to address the challenges posed by the imbalanced class labels in our dataset.

Despite these advancements, a notable gap exists in the application of GCNs for issue classification. GCN, known for its efficacy in graph-structured data (Yao, Mao, and Luo 2019), offers a novel perspective in analyzing interconnected issues. Unlike traditional machine learning models that treat data points independently, GCN can leverage the relational information inherent in issue tracking systems. This is particularly relevant given the complex, networked nature of software development projects where issues are often interrelated. Our study aims to harness this potential of GCN. By applying GCN to the TAOWS dataset (Tawosi et al. 2022), we not only explore a new method of issue classification but also address the limitations of previous models that do not fully utilize the relational dynamics of issue data.

Methodology

In our research, we followed a comprehensive methodology involving four key stages: Dataset Acquisition, Data Preprocessing, Graph Construction, and Model Training and Evaluation. Each step is crucial in developing a GCN model for our study. Below, we detail these stages, showcasing the systematic approach we employed. Furthermore, Figure 1 presents an overview of our methodology.

Dataset Acquisition

Our methodology begins with extracting data from the TAWOS dataset version 1.0 (Tawosi et al. 2022) , comprising open-source Agile software projects from Jira repositories. We used SQL queries to selectively extract key issue attributes such as Title, Description, and Type. The focus was on the three most common issue categories in repositories like MongoDB, Apache, and Mulesoft, identified as Bug, Improvement, and Task, totaling 58,861 issues. This dataset's diversity is crucial for developing a robust GCN model applicable to real-world software project management scenarios. To provide a clearer understanding of our data foundation, Table 1 outlines the distribution of these issues across each of the identified categories.

Table 1: Distribution of Issues Across Repositories and Categories

Repository	Project Name	Category	Issues
MongoDB	SERVER	Bug	20829
		Improvement	9957
		Task	8403
Apache	MESOS	Bug	4706
		Improvement	2331
		Task	1642
Mulesoft	MULE	Bug	5264
		Improvement	3004
		Task	2725
Total number of issues			58,861

Data Pre-processing

Data preprocessing is a crucial step in preparing the dataset for graph construction and subsequent analysis. Our process

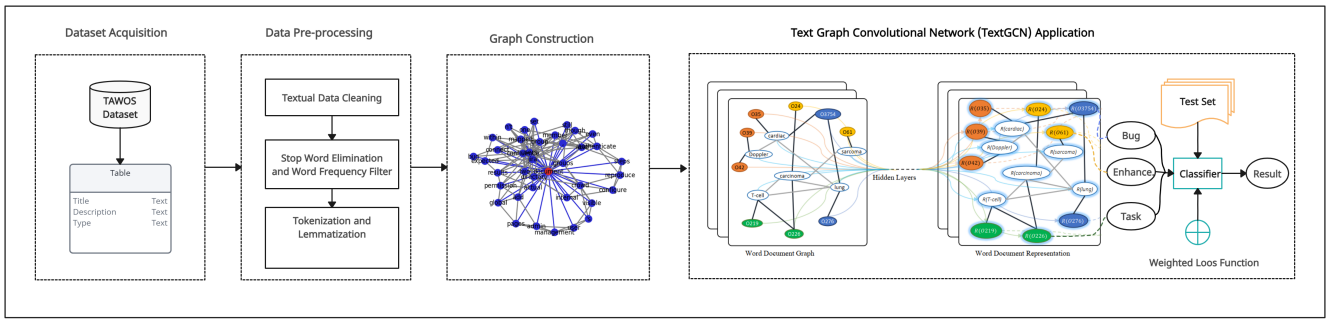

Figure 1: Overview of the GCN-Based Issues Classification Methodology. The process begins with dataset acquisition, followed by data preprocessing, graph construction, and the application of a Text Graph Convolutional Network (TextGCN).

involved several key steps:

Textual Data Cleaning In the initial phase of data preprocessing, we focused on cleaning the textual data. This involved using regular expressions to remove web URLs, special characters, punctuation, and numeric characters from the text. By eliminating these elements, we ensured that the dataset contained only relevant textual content, free from non-standard characters and irrelevant numbers, thereby reducing potential noise and improving the quality of the data for analysis.

Stop Word Elimination and Word Frequency Filter We removed common words (stop words) that are frequent but hold little analytical value and excluded words that appeared less than five times. This approach focused our analysis on more meaningful terms.

Tokenization and Lemmatization Utilizing the NLTK library, we broke down the text of issue reports into individual words or tokens and then lemmatized them to their base forms. This standardizes the text for consistency in analysis.

Graph Construction Method

Our approach constructs a large heterogeneous graph from the entire corpus, similar to the method described in (Fang et al. 2021). The graph comprises nodes representing both documents and words, enabling the capture of high-order neighborhood information crucial for our classification task.

Graph Definition We define a graph $G = (V, E)$, where V represents the set of nodes, including both words and documents in our corpus, and E denotes the set of edges between these nodes.

Node Representation The feature matrix X is set as an identity matrix I, where each word or document is represented as a one-hot vector. This approach simplifies the representation while retaining the essential characteristics of each node.

Edge Construction Edges in the graph are constructed based on two key relationships: Word-Word Co-occurrence and Document-Word Frequency. For word-word edges, we calculate the Pointwise Mutual Information (PMI) between words across the corpus. We also employ a fixed-size sliding window on each document to capture word co-occurrence frequencies globally. For Document-Word Frequency, we use the Term Frequency-Inverse Document Frequency (TF-IDF) to determine the weight of the edge between a document node and a word node.

Text Graph Convolutional Network (TextGCN) Application

Once the graph is constructed, we apply a two-layer GCN model. The second layer's node (word/document) embeddings are of the same size as the label set and are input into a softmax classifier.

$$Z = \text{softmax}(\tilde{A}\text{ReLU}(\tilde{A}XW_0))W_1 \tag{1}$$

where $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, and $softmax(x_i) = \frac{\exp(x_i)}{Z}$ with $Z = \sum_i \exp(x_i)$. where x_i represents one of the three possible output categories (namely, Bug, Enhancement, or Task). The loss function is defined as the cross-entropy error over all labeled documents:

$$L = -\sum_{d \in Y_D} \sum_{f=1}^{F} Y_{df} \ln Z_{df} W_f \tag{2}$$

where Y_D indicates labeled document indices, F represents the number of output labels (which is equal to 3), and Y is the matrix indicating labels, and W_f is the weight of this label. The weight parameter is defined as

$$W_f = \begin{cases} 0, & \text{if } N_f = 0, \\ \frac{\max(N_i)}{N_f}, & \text{if } N_f \neq 0,\ i \subseteq F \end{cases} \tag{3}$$

where N_f is the number of times label f appears in the data. The weight parameter W_f is defined differently depending on whether N_f is zero or non-zero, ensuring appropriate weighting based on the frequency of the label.

Experiment Setting

In this section, we describe our experimental setup in detail.

Evaluation Measure

To assess the model's accuracy and effectiveness and analyze results of our experiments, we choose the F-measure (F%) as our evaluation measures. Precision (P%), recall (R%), and F-measure for a class C_i can be defined as:

$$P(C_i) = \frac{TP}{TP + FP} \tag{4}$$

$$R(C_i) = \frac{TP}{TP + FN} \tag{5}$$

$$F(C_i) = \frac{2 \times P \times R}{P + R} \tag{6}$$

TP (True Positives) refer to the count of issues that the model correctly predicts as belonging to a specific class-label C_i. FP (False Positives) are instances where the model incorrectly predicts issues as belonging to class-label C_i when they do not actually belong to that class. False Negatives (FN) occur when issues that genuinely belong to class-label C_i are overlooked or not recognized by the model, representing another form of classification.

Parameter Settings

For TextGCN in issues classification, we set the embedding size of the convolution layer as 200 and set the window size as 15. While exploring alternative settings to find the optimal value of the parameter, we observed that minor adjustments had a slight impact on the results. Additionally, we set a learning rate of 0.02 and a dropout rate of 0.5. The training process for the TextGCN was at a maximum of 100 epochs. Notably, following the findings of Yao et al. (Yao, Mao, and Luo 2019), we adopted their recommended settings for these two parameters—the learning rate and dropout rate.

Parameter Sensitivity When experimenting with different sliding window sizes (10, 15, 20), this reveals a minor initial rise in test accuracy with an increase in window size. but the average accuracy stops increasing when the window size is larger than 15. as demonstrated in (Yao, Mao, and Luo 2019). However, when assessing the classification performance with different dimensions of the-first layer embeddings (200, 250, 300), the results reveal a negligible improvement in test accuracy with an increase in embedding size.

Experiments Results and Discussion

Answer to RQ1: Effectiveness Evaluation

To address RQ1, various ratios of the training dataset (ratios = 0.6, 0.5, 0.4, 0.3, 0.2 and 0.1) were employed to analyze how these variations influenced model performance. The analysis of the results in Table 2 presents a performance comparison for different training ratios where C_i indicate the class labels. Especially, WAvg denotes weighted average. In this context, the weight assigned to each class label is determined by dividing the number of occurrences of that class label by the total number of occurrences of all class labels.

Our results revealed average F-measure values of 66.31%, 64.69%, 64.47%, 64.8%, 63.71%, and 62.28% for various ratios of the training dataset. We observed a decrease in the average F-measure value as the ratio of the training dataset decreased. Notably, TextGCN in issues classification consistently demonstrated robust performance, achieving a reasonable average F-measure of 62.28% at a 10% training dataset ratio. These findings align with (Yao, Mao, and Luo 2019), emphasizing the efficacy of GCN in low-label-rate scenarios.Furthermore, the performance of the 'Bug' class was particularly noteworthy. It achieved an F-measure of 79.6% at a training ratio of 0.6, which is significantly higher than the average performance across other classes. This suggests that the features of the 'Bug' class are well-captured by the model at this training ratio, indicating a strong model fit for this category. Such a high F-measure underscores the potential of the GCN model to effectively and accurately identify critical issues within agile software development projects, which can substantially enhance the efficiency of the debugging process.

Pre-processing steps including tokenization and lemmatization, significantly contributed to the performance improvement. These steps resulted in a notable average F-measure increase from 43% without pre-processing to 66% with pre-processing, underscoring the importance of effective data pre-processing in enhancing model outcomes.

However, a major limitation is the transductive nature of the GCN model, particularly in the context of text classification where new documents are continuously being generated. Transductive models require retraining with the entire graph, which includes test document nodes (without labels) in the training process to make predictions on unseen documents.

Answer to RQ2: Inductive Learning Approaches

In addressing the second research question, we conducted experiments to assess the efficacy of inductive learning approaches, particularly focusing on the Fast_Text_GCN (Cai et al. 2022), compared to the traditional TextGCN model. For the Fast_Text_GCN models, we employed the default parameter settings as presented in their original paper or implementation, whereas for TextGCN, we adhered to the parameter settings used in the first experiment. We utilized a sample from our dataset, consisting of approximately 24,000 issues exclusively from the SERVER project within the MangoDB repository, ensuring that the volume of data was representative of a real-world agile software development environment. The performance metrics - Precision (P%), Recall (R%), and F1-Score (F%) - were used as indicators of effectiveness are presented in Table 3.

However, the Fast_Text_GCN model exhibited its strengths in the 'Task' class achieving a precision of 70.49% which is higher than TextGCN. This higher precision indicates that Fast_Text_GCN is particularly effective at accurately identifying 'Task' related issues, suggesting a higher level of discernment in classifying such issues. This is a crucial attribute, especially in agile development environments where tasks are frequently updated and prioritized. Despite its lower recall, which indicates a

Table 2: Performance comparison for Different Training Ratios (C1, Bug), (C2, Improvement), (C3, Task).

	Ratio 0.1			**Ratio 0.2**			**Ratio 0.3**			**Ratio 0.4**			**Ratio 0.5**			**Ratio 0.6**		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
C1	80.70	73.52	**76.94**	81.83	73.95	**77.69**	82.78	75.09	**78.75**	83.04	75.05	**78.84**	82	75.42	**78.62**	82.81	76.74	**79.66**
C2	50.67	56.56	53.46	53.05	57.11	55	54.62	57.27	55.92	54.36	60.14	57	53.69	58.74	56	56.55	59.20	57.84
C3	54.47	58.57	56.45	54.83	62.54	58.43	55.49	64.71	59.74	57.61	63.64	60.47	57.08	61.84	59.36	57.61	64.58	60.90
WAvg	61.95	62.89	62.28	63.24	64.53	63.71	64.3	65.69	64.8	65	66.28	65.47	64.29	65.33	64.69	65.66	66.84	**66.13**

Table 3: Performance comparison for TextGCN and Fast_Text_GCN.

	Text_GCN			**Fast_Text_GCN**		
	P(%)	R(%)	F(%)	P(%)	R(%)	F(%)
C1	83.96	74.74	79.08	63.41	69.47	66.30
C2	55.52	63.09	59.07	55.96	66.50	60.77
C3	58.89	65.41%	61.98	70.49	50.47	58.82
WAvg	66.12	67.75	**66.71**	63.28	62.15	**61.97**

propensity to miss some 'Task' instances, the precision of Fast_Text_GCN in correctly classifying tasks that it does identify is indicative of its robustness in dealing with clear-cut cases.

The weighted average (WAvg) performance metrics across all issue classes suggest that the Fast_Text_GCN model performs competitively, albeit slightly trailing behind the traditional model. However, the advantages of Fast_Text_GCN become more apparent when considering the dynamic nature of agile development, where issues continuously evolve and new issues are introduced. The ability of Fast_Text_GCN to generalize to unseen data without retraining on the entire graph is especially pertinent in such a rapidly changing environment, where the agility in updating issue classifications is paramount.

Conclusion

This paper set out to explore the applicability and efficacy of GCNs in the domain of issue classification within agile software development projects. Our findings reveal that GCN models, particularly TextGCN, are highly effective in classifying issues with substantial accuracy, even under varying training dataset ratios. The model's remarkable performance in the 'Bug' class underscores its practical utility in agile environments where timely and accurate bug identification is crucial. For RQ2, the Fast Text GCN model, with its inductive learning capabilities, showed promise in scenarios requiring quick adaptation to new, unseen data. While it did not surpass TextGCN in traditional performance metrics, its agility and adaptability align well with the agile principles of rapid response and flexibility.

Our research opens several avenues for future investigation, aiming to further contextualize and enhance the contributions of our work. Future investigations could involve conducting a more extensive comparison with a wider array of BERT derivatives or other state-of-the-art models. Exploring the potential for future work to compare our model's approach using GCN networks with traditional, non-machine learning methods that utilize centrality measures for modeling and prediction could offer insights into the unique benefits of machine learning-based approaches over conventional methods. Additionally, future research endeavors could explore additional applications or domains where our model could be applied, beyond agile software development. For instance, our model could be utilized in software maintenance tasks such as identifying and prioritizing code refactoring opportunities or predicting software defect severity. Exploring such applications would highlight the adaptability and potential impact of our model in addressing diverse challenges within software engineering

References

Alhindi, W.; Aleid, A.; Jenhani, I.; and Mkaouer, M. W. 2023. Issue-Labeler: an ALBERT-based Jira Plugin for Issue Classification. In *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 40–43.

Cai, H.; Lv, S.; Lu, G.; and Li, T. 2022. Graph convolutional networks for fast text classification. In *2022 4th International Conference on Natural Language Processing (ICNLP)*, 420–425.

Fang, S.; Tan, Y.-s.; Zhang, T.; Xu, Z.; and Liu, H. 2021. Effective prediction of bug-fixing priority via weighted graph convolutional networks. *IEEE Transactions on Reliability* 70(2):563–574.

Huang, Q.; Xia, X.; Lo, D.; and Murphy, G. C. 2020. Automating Intention Mining. *IEEE Transactions on Software Engineering* 46(10):1098–1119. Conference Name: IEEE Transactions on Software Engineering.

Izadi, M.; Akbari, K.; and Heydarnoori, A. 2022. Predicting the objective and priority of issue reports in software repositories. *Empirical Software Engineering* 27(2):50.

Kallis, R.; Di Sorbo, A.; Canfora, G.; and Panichella, S. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 406–409. ISSN: 2576-3148.

Rossi, A.; Tiezzi, M.; Dimitri, G. M.; Bianchini, M.; Maggini, M.; and Scarselli, F. 2018. Inductive–transductive learning with graph neural networks. In *Artificial Neural Networks in Pattern Recognition: 8th IAPR TC3 Workshop, ANNPR 2018, Siena, Italy, September 19–21, 2018, Proceedings 8*, 201–212. Springer.

Siddiq, M. L., and Santos, J. C. S. 2022. BERT-based GitHub issue report classification. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*, 33–36. Pittsburgh Pennsylvania: ACM.

Tawosi, V.; Al-Subaihin, A.; Moussa, R.; and Sarro, F. 2022.

A versatile dataset of agile open source software projects. In *Proceedings of the 19th International Conference on Mining Software Repositories*, 707–711.

Yao, L.; Mao, C.; and Luo, Y. 2019. Graph convolutional networks for text classification. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19, 7370–7377. Honolulu, Hawaii, USA: AAAI Press.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI open* 1:57–81.