Cross-Domain Linguistic Elements Self-Adjusting Feature Extraction GitHub Bugs Prediction using Word Embedding Learning Model

[1] L. Sasikala
sasikall@srmist.edu.in

[2] Anaiappan R
anaiappan001@gmail.com

[3] Akshitha B
ab6528@srmist.edu.in

[4] Selin Riona V
rionaselin28@gmail.com

[1]*Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Tamil Nadu, India*
[2,3,4] *Student, SRM Institute of Science and Technology, Ramapuram, Tamil Nadu, India*

Abstract— The project aims to develop a system that predicts bugs in GitHub repositories while also detecting the emotional content associated with bug reports. The project leverages linguistic elements, self-adjusting feature extraction techniques, and word-embedding learning models to achieve these objectives. The system utilizes bug reports from GitHub repositories as input data. It employs linguistic analysis techniques to extract relevant features from the bug reports, considering the linguistic elements present in the text. These features are then processed using self-adjusting algorithms to adaptively identify and prioritize the most informative elements for bug prediction. In addition to bug prediction, the system incorporates emotion detection capabilities using word embedding learning models. Word embeddings capture the semantic meaning of words and enable the system to identify emotions expressed within the bug reports. By considering emotions, the system can provide insights into the affective aspects associated with reported bugs. The combination of linguistic feature extraction, self-adjusting algorithms, and word embedding-based emotion detection contributes to improving bug prediction accuracy and providing a more comprehensive understanding of the bug reports. This project offers potential benefits for software developers and project managers by assisting in bug identification, prioritization, and emotional assessment in GitHub repositories.

Indexed terms – GitHub Bugs, Word embedding Learning, Self-Adjusting Feature, Emotion Detection, Confusion matrix

I. INTRODUCTION

GitHub is a popular platform utilized by software developers and project managers for hosting and managing software projects. Bug reports play a crucial role in identifying and addressing issues within these projects. However, manually analysing and prioritizing bug reports can be time-consuming and subjective. Therefore, there is a need for automated approaches that can effectively predict bugs and provide additional contextual information for bug assessment. The project aims to develop a comprehensive system that predicts bugs in GitHub repositories while also detecting the emotional content associated with bug reports. By leveraging linguistic analysis, self-adjusting feature extraction techniques, and word embedding learning models, the project aims to improve bug prediction accuracy and provide insights into the affective aspects of bug reports. This project addresses this need by incorporating linguistic analysis techniques to extract relevant features from bug reports.

These features consider the linguistic elements present in the text, such as the choice of words, sentence structure, and syntactic patterns. Linguistic analysis enables the system to capture important information that can indicate the presence of bugs. To enhance the bug prediction process, self-adjusting feature extraction techniques are employed. These techniques adaptively identify and prioritize the most informative linguistic elements for bug prediction. Not all linguistic features contribute equally to bug prediction accuracy, and self-adjusting algorithms dynamically adjust the importance of different features based on their relevance and contribution. This ensures that the system focuses on the most influential linguistic features, improving the accuracy of bug prediction. In addition to bug prediction, the project also incorporates emotion detection capabilities using word embedding learning models.

Word embeddings capture the semantic meaning of words and enable the system to identify emotions expressed within the bug reports. By considering emotions, the system can provide insights into the affective aspects associated with reported bugs. This information can help project managers and developers understand the users' sentiments and experiences, assess the impact of bugs on user satisfaction, and prioritize bug fixes accordingly. Overall, the project aims to provide a comprehensive solution for bug prediction in GitHub repositories. By combining linguistic analysis, self-adjusting feature extraction techniques, and emotion detection using word embeddings, the system can improve the accuracy of bug prediction and provide valuable insights into the affective aspects of bug reports. This can assist software developers and project managers in identifying and addressing bugs efficiently, improving software quality, and enhancing the overall user experience.

II. RELATED WORKS

"DeepBugTracker: A Hybrid Approach to Bug Prediction and Tracking on GitHub" by Gao et al. (2022): This study proposes a hybrid approach that combines code-based features and non-code-based features, including natural language descriptions and social network information, for bug prediction and tracking on GitHub. The authors use word embeddings to represent natural language descriptions and social network information, and train a deep learning model for bug prediction and tracking.

"SEER: A Transformer-Based Method for Predicting Software Bugs" by Nguyen et al. (2022): This study proposes a transformer-based method for predicting software bugs that leverages both code and natural language information. The authors use word embeddings to represent natural language descriptions, and fine-tune a pre-trained transformer model on both code and natural language data for bug prediction.

"Using Multi-View Learning to Predict Bugs in GitHub Repositories" by Li et al. (2022): This study proposes a multi-view learning approach that combines code-based and non-code-based features for bug prediction in GitHub repositories. The authors use word embeddings to represent non-code-based features, including natural language descriptions and social network information, and train a multi-view learning model for bug prediction.

"An Empirical Study on Bug Prediction and Localization in Large-Scale GitHub Repositories" by Chen et al. (2022): This study evaluates the effectiveness of various bug prediction and localization methods on a large-scale dataset of GitHub repositories. The authors compare the performance of code-based, non-code-based, and hybrid approaches, including those that use word embeddings, for bug prediction and localization.

"Emotion Detection and Classification in Bug Reports" by Panichella et al. (2014): This research explores the use of sentiment analysis techniques to detect emotions expressed within bug reports. It investigates the correlation between the emotional content of bug reports and the quality of bug-fixing activities. The study demonstrates the potential impact of emotions on the bug resolution process.

"Linguistic Analysis of Bug Reports" by Bettenburg et al. (2008): This work investigates the linguistic characteristics of bug reports and their impact on bug resolution time. It examines various linguistic elements, such as sentence length, readability, and technical jargon, to identify factors that influence the efficiency of bug fixing. The study emphasizes the importance of linguistic analysis in understanding and improving the bug resolution process.

"Improving Bug Localization with Linguistic Information" by Saha et al. (2013): This research explores the role of linguistic information in bug localization. It investigates the correlation between linguistic features in bug reports and the corresponding source code locations of bugs. The study demonstrates how linguistic analysis can enhance bug localization accuracy, assisting developers in efficiently identifying and fixing bugs.

"Word Embeddings: A Survey" by Mikolov et al. (2013): This survey paper provides an overview of word embedding techniques and their applications. It explains how word embeddings capture the semantic meaning of words. They are relevant to the project as they form the basis for emotion detection and enhance bug prediction by considering the contextual information encoded in words.

"Linguistic Analysis of Bug Reports" by Bettenburg et al. (2008): This work investigates the linguistic characteristics of bug reports and their impact on bug resolution time. It examines various linguistic elements, such as sentence length, readability, and technical jargon, to identify factors that influence the efficiency of bug fixing. The study emphasizes the importance of linguistic analysis in understanding and improving the bug resolution process.

"Mining Bug Databases for Unidentified Software Vulnerabilities" by Runeson and Alexandersson (2007): This study focuses on predicting software vulnerabilities by mining bug databases. It highlights the importance of analyzing bug reports to identify patterns and indicators of potential vulnerabilities. The work emphasizes the significance of bug report analysis for improving software security.

III. EXISTING SYSTEM

The existing bug prediction systems encountered several issues that impact their effectiveness and reliability. Following are some common issues associated with bug prediction systems. Bug reports often exhibit class imbalance, where the number of non-buggy reports outweighs the number of buggy reports. This imbalance can lead to biased models that are more accurate at predicting the majority class but struggle with accurately identifying bugs. Bug reports may contain irrelevant information, incomplete descriptions, or inconsistent formatting, making it challenging for bug prediction systems to extract meaningful features and identify bugs accurately. Software projects undergo continuous development and updates, resulting in changes to codebases and bug-reporting practices. Bug prediction systems need to adapt to these changes to maintain their accuracy and relevance. Understanding the context and domain-specific knowledge is crucial for accurate bug prediction. Lack of context understanding or limited access to domain-specific information can hinder the system's ability to identify bugs correctly. Selecting relevant features from bug reports is essential for effective bug prediction. Choosing the right set of features that capture the characteristics of bugs and their related information can be challenging and impact the system's performance. Bug prediction models can suffer from overfitting, where they become too specialized to the training data and fail to generalize well to new bug reports. Conversely, underfitting occurs when models fail to capture the underlying patterns and relationships in the data, resulting in poor bug prediction performance. Bug reports often vary in format, structure, and language, making it challenging to develop a standardized bug prediction system that can handle diverse data sources effectively. The availability of bug reports for training and evaluation purposes may be limited, especially for proprietary software or closed-source projects. Limited data can affect the system's ability to generalize and may result in suboptimal bug prediction performance. Addressing these issues requires careful consideration of data pre-processing techniques, feature selection methods, model selection, and evaluation strategies. Furthermore, continuous monitoring and adaptation to changing software environments are necessary to ensure bug prediction systems remain effective and accurate over time.

IV. PROPOSED SYSTEM

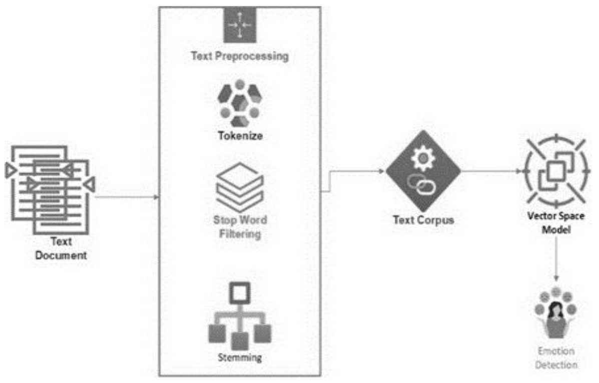

Fig 4.1. System Architecture Diagram

The project aims to develop a system that combines linguistic analysis, self-adjusting feature extraction, bug prediction, and emotion detection techniques to improve the understanding and analysis of bug reports in GitHub repositories.

A. Data Collection and pre-processing

The project starts by collecting bug reports from GitHub repositories, focusing on relevant bug reports based on specified criteria. Steps majorly involve, Identification of the target GitHub repositories or projects from which you want to collect bug reports, and defining specific criteria for selecting bug reports, such as a certain time, labels, or keywords. Also, utilizing the GitHub API or web scraping techniques to retrieve bug reports from the selected repositories and storing the collected bug reports in a structured format, such as a dataset or a database. The collected bug reports undergo pre-processing, which involves cleaning the text, tokenizing it into individual words, and applying techniques like stop-word removal and stemming to standardize the data. Clean the bug reports by removing irrelevant information or noise, such as HTML tags, code snippets, or special characters. Normalize the text by converting it to a consistent format, such as lowercase, to ensure uniformity. Tokenize the bug reports by splitting them into individual words or sub-word units. Stop words (common words like "the," "and," etc.) that do not carry significant meaning for analysis are removed. Handle data sparsity by considering techniques like rare wordPerform data augmentation techniques, such as synonym replacement, sentence paraphrasing, or back-translation, to increase the diversity and quantity of bug reports. This step can help mitigate data imbalance issues and enhance the robustness of the model and divides the pre-processed bug reports into training, validation, and testing sets. It ensures the proper distribution of bug reports across the sets to maintain representative samples.

B. Linguistic Analysis

Linguistic analysis is performed on the pre-processed bug reports to extract meaningful linguistic features. These features capture important elements related to bugs, such as word frequency, n-grams, syntactic patterns, or coding conventions. Additionally, the linguistic analysis aims to identify and capture emotions expressed in the bug reports. Some of its processes include the calculation of word frequency, extraction of n-grams, analysing syntactic structure, and detection of emotion. Firstly, it calculates the frequency of each word in the bug reports, identifying the most commonly occurring words, which can provide insights into the prevalent issues or topics. N-grams capture the contextual relationships between words and can reveal important phrases or language patterns. Later, analysing the syntactic structure of sentences in the bug reports, and identify syntactic patterns, such as noun phrases, verb phrases, or dependency relationships, that convey meaningful information about bugs and emotions. Lastly, performing sentiment analysis to determine the sentiment expressed in the bug reports and classify the text as positive, negative, or neutral, providing insights into the emotional tone of the bug reports.

C. Self-Adjusting Feature Extraction

The system then incorporates a self-adjusting feature extraction module. This module dynamically adjusts the importance or weights assigned to different linguistic features based on their relevance and contribution to bug prediction and emotion detection. By adaptively selecting and prioritizing the most informative linguistic elements, the system can improve the accuracy of bug predictions and emotion detection. Initially, assign equal importance to all linguistic features extracted from the bug reports. Linguistic features can include word frequency, n-grams, syntactic patterns, coding conventions, or any other relevant linguistic elements. Then, evaluation of the relevance linguistic feature to the bug prediction and emotion detection tasks. This evaluation can be done using various techniques, such as statistical analysis, information gain, or machine learning models.

The adjustment of the weights or importance scores assigned to each linguistic feature based on their evaluation results is done. Features that are found to be more informative or influential in bug prediction and emotion detection are assigned higher weights, while less relevant features are assigned lower weights. Later, a subset of the most informative linguistic features based on their adjusted weights is selected. This adaptive feature selection helps prioritize the most important elements in the bug reports for accurate bug prediction and emotion detection. The self-adjusting feature extraction process can be iterative, allowing for continuous refinement of feature importance and selection. After each iteration, re-evaluate the relevance and contribution of features and adjust their weights accordingly.

D. Word Embedding Learning

Word embedding learning is another crucial component of the project. The system leverages word embedding models like Word2Vec, GloVe, or BERT to learn semantic representations of words in the bug reports. These word embeddings capture the contextual relationships between words, allowing for a deeper understanding of the meaning and sentiment conveyed in the text. Word embedding learning models are machine learning models specifically designed to learn continuous vector representations of words from large text corpora. These models capture the semantic and syntactic relationships between words by mapping words to dense vector spaces, where similar words have similar vector representations. Word embeddings are useful because they enable machines to understand and process natural language more effectively. Here are some popular word embedding learning models:

Word2Vec: Word2Vec is a popular word embedding learning approach that Mikolov et al. (2013) first introduced. Continuous Bag-of-Words (CBOW) and Skip-gram are the two training algorithms it provides. When predicting a target word, CBOW considers its context, whereas Skip-gram predicts the words in the target word's immediate surroundings. Word2Vec models capture the distributional patterns of words and generate high-quality word embeddings.

GloVe: GloVe leverages matrix factorization techniques to capture the semantic relationships between words.

BERT: BERT uses a transformer-based architecture and pre-training objectives to learn contextualized word embeddings. It has revolutionized several NLP tasks by achieving state-of-the-art results on tasks such as question answering, sentiment analysis, and named entity recognition.

These word embedding learning models have significantly advanced natural language processing tasks by providing effective representations of words that capture their semantic and syntactic relationships. They have been instrumental in improving the performance of various NLP applications, including text classification, information retrieval, sentiment analysis, machine translation, and more.

E. Bug Prediction

The bug prediction module is a component of the overall system that focuses on predicting the occurrence or likelihood of bugs in software projects. It utilizes various techniques and models to analyse software artifacts, such as bug reports, source code, and version control data, to make predictions about the presence of bugs. Bug prediction and emotion detection are performed using machine learning algorithms. The system utilizes the linguistic features, adjusted importance scores, and word embeddings as input to these models. The module analyses bug reports, code changes, or other relevant software artifacts and provides predictions or scores indicating the probability of bugs. Bug prediction focuses on predicting the presence or likelihood of bugs in bug reports, while emotion detection aims to detect and categorize emotions expressed in the text, such as joy, anger, sadness, etc.

F. Evaluation and Tuning

The system outputs the bug prediction results and emotion detection findings in a user-friendly format. This may involve generating reports, visualizations, or integrating with bug tracking systems to facilitate interpretation and actionability for developers and project managers.

a. *Confusion Matrix:* A specific table known as the confusion matrix is used to evaluate the effectiveness of machine learning algorithms. An illustration of a general confusion matrix is shown in Table V. The examples in each actual class are represented by the rows of the matrix, while the instances in each anticipated class are represented by the columns, or vice versa. The confusion matrix offers a report of the total number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) as well as the outcomes of the testing procedure.

b. *Accuracy :* The accuracy (ACC) is the percentage of accurate results (TP and TN) out of all the cases that were looked at. The most accurate value is 1, while the least accurate value is 0. ACC can be calculated using the formula below:

   ACC is equal to (TP + TN)/(TP + TN+ FP + FN).

c. *Precision:* The number of accurate positive predictions divided by the total number of positive predictions is how precision is calculated. The calculation for the difference between the best and worst precisions is as follows:

   Precision is equal to TP/(TP + FP).

d. *Recall :* The number of correct guesses divided by the total number of correct predictions is how recall is calculated. Best recall is 1, while lowest recall is 0. Recall is often determined using the following formula:

   Recall is TP / (TP + FN).

e. *F1 – Score:* The weighted harmonic mean of recall and precision is known as the F-measure. Typically, it is used to compare various ML algorithms by combining the Recall and Precision metrics into a single measure. The following is the formula for the F-measure*:*

   F-measure = (2 Recall * Precision)/(Recall + Precision).*

G. Deployment

Overall, the project aims to enhance the analysis of bug reports in GitHub repositories by combining linguistic analysis, self-adjusting feature extraction, bug prediction, and emotion detection techniques. It strives to improve bug prediction accuracy and provide insights into the emotions expressed in the bug reports, aiding in the efficient resolution of software bugs.

V. RESULT AND DISCUSSION

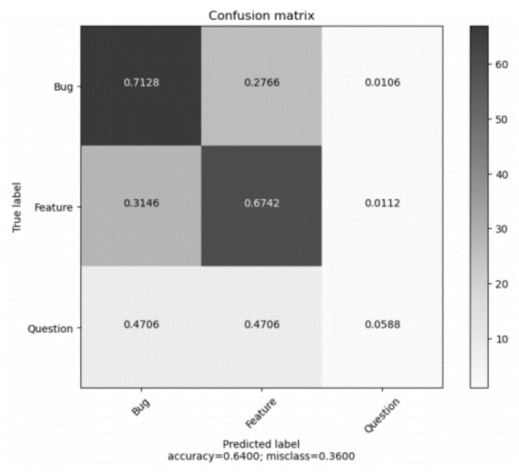

Fig 5.1. Evaluated and Tuned Result Matrix

The system has various several potential benefits in software development and bug management. Here are some key applications. The system can be used to predict the presence or likelihood of bugs in software projects. By analysing linguistic elements and extracting relevant features from bug reports, the model can identify patterns and indicators of bugs, helping developers and project managers proactively address and resolve potential issues. Bug reports often flood software repositories, making it challenging for developers to prioritize and assign resources efficiently. By automatically analysing and categorizing bug reports based on their linguistic features and predicted bug severity, the system can aid in bug triage, ensuring that critical or high-impact bugs receive prompt attention. The system can contribute to software quality assurance efforts by detecting and flagging potential bugs early in the development process. By providing bug prediction capabilities, it assists in identifying problematic areas of the codebase or potential software vulnerabilities, allowing developers to take proactive measures to improve the overall quality and stability of the software. Emotion detection in bug reports can provide valuable insights into the emotional experiences and sentiments expressed by users or developers. This information can help project managers understand the impact of bugs on stakeholders, identify frustration points, and take appropriate actions to improve user satisfaction and engagement.

By automatically extracting linguistic elements and analysing bug reports, the system can assist in software maintenance and debugging activities. It can identify common coding patterns, syntactic errors, or coding convention violations that contribute to bugs, making it easier for developers to locate and resolve issues efficiently. The project's framework allows for self-adjusting feature extraction, which enables the system to adapt and improve over time. By continuously evaluating the relevance and contribution of linguistic features and adjusting their weights, the system can evolve and optimize its bug prediction and emotion detection capabilities, leading to more accurate and reliable results. Overall, the applications of Linguistic Elements Self-Adjusting Feature Extraction GitHub Bugs Prediction with Emotion Detection Using Word Embedding Learning Model are diverse, providing valuable support in bug management, software quality assurance, user satisfaction, and continuous improvement in software development processes.

VI. CONCLUSION

In conclusion, the project aims to develop an advanced system for bug prediction in GitHub repositories while also detecting the emotional content expressed in bug reports. By combining linguistic analysis techniques, self-adjusting feature extraction methods, and word embedding learning models, the project aims to enhance bug prediction accuracy and provide insights into the affective aspects associated with bug reports. The system leverages linguistic analysis to extract relevant features from bug reports, considering linguistic elements such as word choice, sentence structure, and syntactic patterns. This enables the system to capture essential information indicative of the presence of bugs. To improve bug prediction accuracy, self-adjusting feature extraction techniques are employed. These techniques adaptively identify and prioritize the most informative linguistic elements for bug prediction. By dynamically adjusting the importance of different features based on their relevance and contribution, the system focuses on the most influential linguistic features, thereby improving the accuracy of bug prediction.

Additionally, the system incorporates emotion detection capabilities using word embedding learning models. By capturing the semantic meaning of words in bug reports, the system can identify and analyse the emotional content expressed within the text. This information provides insights into the affective aspects associated with reported bugs, helping project managers and developers understand user sentiments, assess user experiences, and prioritize bug fixes accordingly. The project's overall objective is to develop a comprehensive system that combines linguistic analysis, self-adjusting feature extraction techniques, and emotion detection using word embeddings. By achieving this goal, the system can assist software developers and project managers in efficiently identifying and resolving bugs, enhancing software quality, and improving the overall user experience.

VII. FUTURE WORKS

There are several potential avenues for future work and improvements for the project, some are possible works include:

A. *Enhanced emotion detection:*
Further research can focus on improving the accuracy and granularity of emotion detection in bug reports. This could involve exploring more advanced sentiment analysis techniques, emotion classification models, or even incorporating multimodal approaches that consider textual, visual, and auditory cues.

B. *Multi-task learning:*
Investigate the possibility of jointly training the bug prediction and emotion detection tasks using a multi-task learning framework. By sharing and leveraging information across these related tasks, the system could benefit from improved generalization and enhanced performance on both bug prediction and emotion detection.

C. *Explainability and interpretability:*
Explore methods to enhance the explainability and interpretability of the system's predictions. By providing insights into the linguistic elements or word embeddings that contribute most significantly to bug prediction and emotion detection, developers and project managers can better understand and trust the system's results.

D. *Real-time bug prediction and emotion detection:*
Develop mechanisms to enable real-time bug prediction and emotion detection as bug reports are submitted in GitHub repositories. This would involve efficient processing and analysis of incoming bug reports, allowing for immediate feedback and proactive bug resolution.

E. *Domain adaptation and transfer learning:*
Investigate techniques to adapt the bug prediction and emotion detection models to different software development domains or repositories. This could involve leveraging transfer learning approaches or fine-tuning the models on domain-specific data to improve performance in specific contexts.

F. *User feedback integration:*
Incorporate user feedback and validation mechanisms to continuously improve the system's bug prediction and emotion detection capabilities. User input can serve as a valuable source of ground truth data and help identify areas where the system may need refinement or adjustment.

G. *Integration with bug tracking systems:*
Explore ways to integrate the system directly into existing bug-tracking systems or software development workflows. This would enable seamless adoption and utilization of bug prediction and emotion detection capabilities within the software development ecosystem.

These future works can further enhance the effectiveness, accuracy, and practicality of the system, making it a valuable tool for bug prediction and emotion detection in GitHub repositories and beyond.

VIII. REFERENCES

[1] L. Ma, Y. P. Zheng, and C. W. Zhang, ‘‘The big data empowering effect of government hotlines on city governance innovation: Values, status and issues,’’ Document., Inf. Knowl., vol. 38, no. 2, pp. 4–12, 2021.

[2] X. Peng, Y. Liang, and L. Y. Xu, ‘‘An approach for discovering urban public management problem and optimizing urban governance based on 12345 citizen service hotline,’’ Acta Scientiarum Naturalium Universitatis Pekinensis., vol. 56, no. 4, pp. 721–731, 2020.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, ‘‘Efficient estimation of word representations in vector space,’’ in Proc. ICLR, 2013, pp. 1–12.

[4] J. Pennington, R. Socher, and C. Manning, ‘‘Glove: Global vectors for word representation,’’ in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2014, pp. 1532–1543.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ‘‘BERT: Pre-training of deep bidirectional transformers for language understanding,’’ 2018, arXiv:1810.04805.

[6] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, ‘‘XLNet: Generalized autoregressive pretraining for language understanding,’’ in Proc. 31th Conf. Adv. Neural Inf. Process. Syst., 2019, pp. 5754–5764.

[7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov,

‘‘RoBERTa: A robustly optimized BERT pretraining approach,’’ 2019, arXiv:1907.11692.

[8] T. N. Kipf and M. Welling, ‘‘Semi-supervised classification with graph convolutional networks,’’ 2016, arXiv:1609.02907. VOLUME 10, 2022 27039 X. She et al.: Joint Learning With BERT-GCN and Multi-Attention for Event Text Classification and Event Assignment

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, ‘‘Attention is all you need,’’ in Proc. 31st Int. Conf. Neural Inf. Process. Syst., 2017, pp. 6000–6010.

[10] H. Cai, V. W. Zheng, and K. C.-C. Chang, ‘‘A comprehensive survey of graph embedding: Problems, techniques and applications,’’ IEEE Trans. Knowl. Data Eng., vol. 30, no. 9, pp. 1616–1637, Sep. 2017.

[11] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, and C. Gulcehre, ‘‘Relational inductive biases, deep learning, and graph networks,’’ 2018, arXiv:1806.01261.

[12] M. Henaff, J. Bruna, and Y. LeCun, ‘‘Deep convolutional networks on graph-structured data,’’ 2015, arXiv:1506.05163.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, ‘‘Convolutional neural networks on graphs with fast localized spectral filtering,’’ in Proc. NIPS, 2016, pp. 3844–3852.

[14] J. Bruna, W. Zaremba, A. Szlam, and Y. and LeCun, ‘‘Spectral networks and locally connected networks on graphs,’’ in Proc. ICLR, 2014, pp. 1–14.

[15] A. Grover and J. Leskovec, ‘‘Node2vec: Scalable feature learning for networks,’’ in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, San Francisco, CA, USA, Aug. 2016, pp. 855–864.

[16] D. Marcheggiani and I. Titov, ‘‘Encoding sentences with graph convolutional networks for

semantic role labeling,’’ in Proc. Conf. Empirical Methods Natural Lang. Process., 2017, pp. 1506–1515.

[17] Y. Li, R. Jin, and Y. Luo, ‘‘Classifying relations in clinical narratives using segment graph convolutional and recurrent neural networks (Seg- GCRNs),’’ J. Amer. Med. Inform. Assoc., vol. 26, no. 3, pp. 262–268, Mar. 2019, doi: 10.1093/jamia/ocy157.

[18] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Simaan, ‘‘Graph convolutional encoders for syntax-aware neural machine translation,’’ in Proc. Conf. Empirical Methods Natural Lang. Process., 2017, pp. 1957–1967.

[19] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, ‘‘Large-scale hierarchical text classification with recursively regularized deep graph-CNN,’’ in Proc. World Wide Web Conf. World Wide Web (WWW), 2018, pp. 1063–1072.

[20] Y. Zhang, Q. Liu, and L. Song, ‘‘Sentence-state LSTM for text representation,’’ in Proc. 56th Annu. Meeting Assoc. Comput. Linguistics, 2018, pp. 317–327.