International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Automatic Component Prediction for Issue Reports

DR. Bhargavi Peddi Reddy[1], P. Bindu Sriya[2], Hrishitha Rayapati[3]

[1,2,3]Department of Computer Science, Vasavi College of Engineering, Hyderabad, India

Abstract.
Every day, there's a constant influx of software problems emerging during the testing and maintenance phases. With software becoming larger and more intricate, this issue count is on the rise, necessitating swift management. However, handling these issues manually proves challenging due to their complexity and sheer volume, often leading to inefficient and costly outcomes.
Previous research endeavors have explored automating this triage process through machine learning and word-based language models, aiming to predict the component related to an issue. This component information is crucial for software engineers to pinpoint the problem's location. Yet, existing methods have fallen short of expectations due to their structural limitations and failure to grasp the context of words. To address this, we propose a novel approach leveraging pretrained language models, particularly fine-tuning BERT on a diverse dataset of issue reports. By doing so, we surpass the limitations of LSTM-based methods and enhance performance in predicting issue components.

Keywords: Component recommendation, machine learning, natural language processing, pretrained language model, software engineering.

1 Introduction

The increasing complexity and scale of modern software development have led to a surge in reported issues during testing and maintenance phases. Issue tracking systems like Jira and Bugzilla play a crucial role in managing these issues systematically. However, the manual triage of these issues by human experts has become increasingly challenging due to their volume and complexity.
A problem report usually includes various details like title, description, reporter information, product, component, priority, severity, and additional data. Identifying the appropriate component in a problem report is crucial for software engineers to precisely locate the reported issue or bug. In projects with changing scopes, the number of components may significantly increase over time, complicating manual triage further.
The manual triage process involves assigning issue reports to appropriate components based on expert knowledge of project modules and codebases. However, this process is labor-intensive and prone to errors, resulting in significant time and cost expenditures. For instance, in real-world industrial settings like the Eclipse project, a quarter of issue reports were reassigned due to triage errors, costing substantial person-hours daily.
To address these challenges, there is a compelling need to automate and optimize the issue triage process using advanced tool support. This paper introduces an innovative method utilizing fine-tuned pretrained

models to forecast components from issue report data. Through the utilization of pretrained language models, we seek to address the structural constraints identified in LSTM-based approaches.

In this study, we evaluate the effectiveness of fine-tuned pretrained language models by comparing their performance against a baseline method. Our goal is to demonstrate the superiority of pretrained language models in accurately predicting issue components, thereby paving the way for efficient automation of the issue triage process and improving overall software quality.

2 Literature Survey

Sureka et al. endeavored to forecast components utilizing a machine learning approach comprising term frequency-inverse document frequency along with a component reassignment graph. The TF-IDF technique facilitates the extraction of content-oriented textual features. During experimentation, they observed that component reassignment led to a decline in accuracy. Consequently, they integrated a component reassignment graph that accounts for alterations in the issue report. Thus, they devised a predictive model that merges TF-IDF with a component reassignment graph to generate the top-k outcomes for components.

Yan et al. introduced a discriminative probability latent semantic analysis model aimed at predicting components based on the issue report's topic. They concentrated on identifying which component closely aligns with the terms describing its function within the bug report. To achieve this, they developed a semantic analysis model where issue reports served as documents and components as categories, following the approach suggested by Lu et al. Their experiment involved extracting the top-k results from a dataset comprising 6,000 issue reports' titles and descriptions, covering ten components. They attained enhanced recall outcomes at k, signifying the top-k component predictions. However, their dataset comprised only ten components, fewer than the one we utilized. Thus, Yan et al.'s model showcases the potential of component prediction through semantic analysis, even with a smaller component set compared to other datasets.

3 Design

A. Dataset

The dataset was collected from the Eclipse platform and Bugzilla Firefox, each comprising approximately 10,000 records with an average of 18 components. The dataset is structured into three classes: title, description, and component. We allocated 80% of the dataset for training purposes and the remaining 20% for testing.

Bugzilla Firefox			
Method Name	Task1 Quantity	Task2 Quantity	Task3 Quantity
General	2,805	2,805	390
Address Bar	1,863	1,863	390
Bookmarks & History	1,787	1,787	390
New Tab Page	1,480	1,480	390
Messaging System	1,397	1,397	390
Theme	1,060	1,060	390
Sync	1,010	1,010	390
Preferences	801	801	390
Search	776	776	390
Tabbed Browser	583	583	390
PDF Viewer	581	581	390
Toolbars and Customization	568	568	390
Menus	404	404	390
Protections UI	385	385	385
Installer	367	367	367
Security	366	366	366
Site Identity	359	359	359
about:logins	343	343	343
Session Restore	245	245	245
Untriaged	228	228	228
Enterprise Policies	211	211	211
Downloads Panel	191	191	191
File Handling	179	179	179
WebPayments UI	173	173	173
Site Permissions	141	141	141
Migration	139	139	139
Pocket	129	129	129
Shell Integration	114	114	114
Private Browsing	109	109	109
Type: REG_SZ	106	106	106
Nimbus Desktop Client	99	99	99
Firefox Accounts	91	91	91
Extension Compatibility	89	89	89
Tours	84	84	84
Normandy Client	80	80	80
Top Sites	76	76	76
Keyboard Navigation	67	67	67
Offset	55	55	55
Disability Access	53	53	53
Remote Settings Client	52	52	52
Screenshots	50	50	50
Pioneer	39	-	-
Page Info Window	35	-	-
Normandy Server	26	-	-
Foxfooding	24	-	-
Distributions	15	-	-
Launcher Process	9	-	-
Firefox Monitor	8	-	-
Activity Streams: General	7	-	-
Headless	6	-	-
Components	5	-	-
Total(Num of issues)	**19,870**	**19,696**	**9,651**
Average	**390**		

Eclipse Platform			
Method Name	Task1 Quantity	Task2 Quantity	Task3 Quantity
UI	7,701	7,701	994
Debug	3,289	3,289	994
Ant	1,752	1,752	994
SWT	1,650	1,650	994
Team	929	929	929
Releng	733	733	733
Text	630	630	630
CVS	523	523	523
Compare	356	356	356
IDE	315	315	315
Resources	273	273	273
User Assist	248	248	248
Doc	180	180	180
Search	144	144	144
Runtime	93	93	93
Update (deprecated - use Ecli	57	57	57
PMC	9	-	-
Website	3	-	-
WebDAV	1	-	-
Total(Num of issue)	**18,886**	**18,873**	**8,457**
Average	**994**		

B. Algorithms used

1. **RoBERTa (Transformer-Based Language Model):**

RoBERTa is a transformer-based language model that utilizes bidirectional self-attention mechanisms to understand the context of words and their relationships within sentences. It employs a masked language modeling objective and pre-training on large text corpora to capture complex linguistic patterns.

- The text data is tokenized using the RoBERTa tokenizer, segmenting input sequences into sub word tokens.
- The pre-trained RoBERTa model is fine-tuned on the issue report data to perform multilabel classification for component prediction.
- Adam optimizer with a learning rate of 5e-6 is used for training. The model is trained for a specified number of epochs to optimize component prediction accuracy.

2. LSTM (Word-Based Recurrent Neural Network):

Long Short-Term Memory is a recurrent neural network (RNN) capable of learning long-range dependencies in sequential data. It processes input data word by word and maintains memory across time steps.

- Text data is tokenized into sequences of words and padded to ensure uniform length for batch processing.
- Embedding layers map words into dense vector representations to capture semantic meaning.
- LSTM layers process sequential input data and learn to extract relevant features for component prediction.
- The LSTM model is trained using Adam optimizer with a specified learning rate and optimized over multiple epochs to maximize prediction accuracy.

4 Implementation

1. RoBERTa (Bidirectional Encoder Representations from Transformers)

1. Dataset Loading and Preprocessing

Load the issue report dataset containing titles, descriptions, and corresponding component labels. Preprocess the text data by cleaning, tokenizing, and preparing it for input into RoBERTa.

2. Tokenization

Utilize the RoBERTa tokenizer to tokenize the text sequences. Encode the tokenized sequences and apply necessary padding for uniform length.

3. Model Architecture

Initialize the RoBERTa model for sequence classification.

Add additional layers for multilabel classification on top of the RoBERTa base.

Classification Layers: Implement layers for multilabel classification, such as dense layers with appropriate activation functions.

Output Layer: Use a sigmoid activation function for multilabel classification to predict multiple component labels simultaneously.

4. Training Configuration

Used the Adam optimizer with a specified learning rate.

Set the loss function to binary cross-entropy for multilabel classification.

Defined the number of epochs =15 and batch size =128 for training.

5. Training and Fine-Tuning

Fine-tuned the RoBERTa model on the preprocessed dataset. Trained the model by fitting it to the training data with specified parameters. Monitored training progress, evaluating loss and performance metrics during training.

6. Evaluation

Evaluated the fine-tuned RoBERTa model on the test dataset. We calculated evaluation metrics such as accuracy, precision, recall, and F1-score for multilabel classification.

7. Prediction

We used the trained RoBERTa model to predict component labels for new issue reports.

We implemented a user interface (UI) for inputting new issue reports and displaying predicted component labels.

2. LSTM (Long Short-Term Memory) Model Implementation

1. Data Preparation

We loaded and preprocessed the issue report dataset, which included titles, descriptions, and associated component labels. Text data was preprocessed by cleaning and tokenizing it into sequences suitable for LSTM input.

2. Embedding Layer

We initialized the LSTM model, incorporating an embedding layer to convert text data into numerical representations. The embedding layer mapped each word to a dense vector space, capturing semantic relationships between words.

3. LSTM Layers

We configured the LSTM model with multiple LSTM layers to capture sequential dependencies in the text data. Each LSTM layer processed input sequences, retaining memory over long sequences of words.

4. Dense Layers

Additional dense layers were added after the LSTM layers to perform classification tasks. These layers enabled the model to learn higher-level features and make predictions based on sequential input.

5. Training Setup

We used the Adam optimizer with a specific learning rate to train the LSTM model.

The model was trained using a defined number of epochs and batch size, iterating over the training data.

6. Model Training

We trained the LSTM model on the preprocessed dataset, optimizing its parameters to minimize loss and improve accuracy. During training, we monitored performance metrics such as loss and accuracy to assess model convergence.

7. Evaluation

After training, we evaluated the LSTM model's performance on a held-out test dataset. Evaluation metrics such as accuracy, precision, recall, and F1-score were computed to assess the model's effectiveness.

8. Prediction

Using the trained LSTM model, we made predictions on new issue reports to assign component labels. We developed a prediction pipeline that incorporated the LSTM model for real-time component prediction based on issue report content.

5 Results

We have compared the performances of the models. The results are as follows:

LSTM	ROBERT
58%	74%

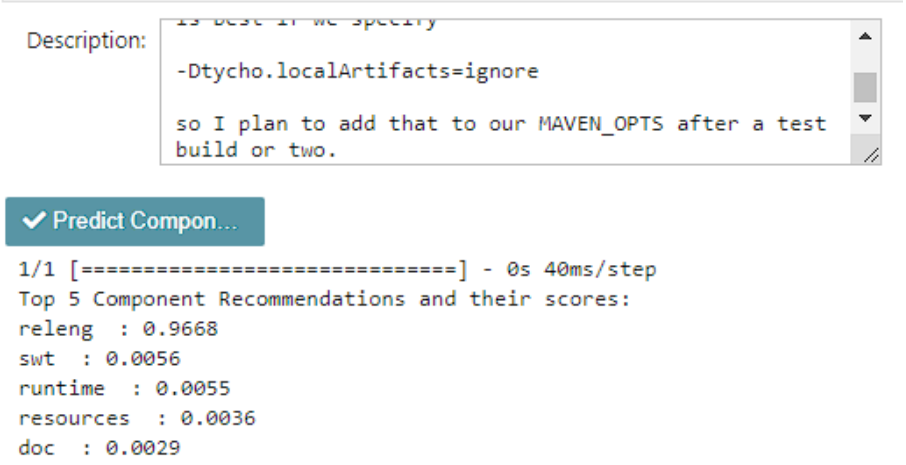

Fig.1. Component Prediction

6 Conclusion & Future Work

In our study, we proposed a technique to enhance automatic component prediction by leveraging pre-trained language models that work at the sentence level, with a specific focus on deep learning frameworks. Earlier methods based on word-level deep learning models showed inadequate performance. To tackle this issue, we fine-tuned RoBERTa to suit a dataset of issue reports covering multiple categories. We assert that utilizing sentence-level pretrained models, which extract information from entire sequences, significantly improved the effectiveness of our approach, yielding superior experimental results compared to the LSTM model.

Our experimental findings reveal the high efficacy of fine-tuned pretrained language models for automatic component prediction. Remarkably, our approach maintained strong accuracy even with smaller datasets, suggesting that performance could be further improved with larger and more detailed datasets in the future. To enhance our method, we aim to incorporate data augmentation techniques into issue reports in future research. Adapting a pretrained language model specifically for issue report datasets has the potential to greatly enhance automatic component prediction. In industrial settings, manual issue triage is often inefficient due to the frequent updates and expansions of software projects, highlighting the importance of applying and evaluating our approach within industrial projects.

Acknowledgement

We thank Vasavi College of Engineering (Autonomous), Hyderabad for the support extended towards this work.

References

1. M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, and A. Ghose, ‘‘Predicting components for issue reports using deep learning with information retrieval,’’ in Proc. 40th Int. Conf. Softw. Eng., Companion Proceeedings, May 2018, pp. 244–245.
2. K. W. Church, ‘‘Word2Vec,’’ Nat. Lang. Eng., vol. 23, no. 1, pp. 155–162, 2017.
3. J. Anvik and G. C. Murphy, ‘‘Reducing the effort of bug report triage: Recommenders for development-oriented decisions,’’ ACM Trans. Softw. Eng. Methodol., vol. 20, no. 3, pp. 1–35, Aug. 2011.
4. S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, ‘‘Applying deep learning based automatic bug triager to industrial projects,’’ in Proc. 11th Joint Meeting Found. Softw. Eng., Aug. 2017, pp. 926–931.
5. W. Zhang, ‘‘Efficient bug triage for industrial environments,’’ in Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME), Sep. 2020, pp. 727–735.
6. M. Yan, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, ‘‘A component recommender for bug reports using discriminative probability latent semantic analysis,’’ Inf. Softw. Technol., vol. 73, pp. 37–51, May 2016.
7. A. Sureka, ‘‘Learning to classify bug reports into components,’’ in Proc. Int. Conf. Model. Techn. Tools Comput. Perform. Eval. Prague, Czech Republic: Springer, 2012, pp. 288–303.