Automatic Detection Method for Software Requirements Text with Language Processing Model

Li Zhang [1]
Hunan University
Changsha, China
zhangli0208@hnu.edu.cn

Liubo Ouyang [2,*]
Hunan University
Changsha, China
oylb@hnu.edu.cn

Jiahao Qin [3]
Hunan University
Changsha, China
694024213@qq.com

2023 6th International Conference on Software Engineering and Computer Science (CSECS) | 979-8-3503-0637-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSECS60003.2023.10428155

***Abstract*—Requirements analysis plays a crucial role in the process of software development. However, due to many factors such as the dichotomy of natural language, requirement texts often present diverse defects, which may lead to anomalies and functional deviations in software systems. In order to solve the above challenges, our study proposes a deep learning-based defect detection method for requirement text, which utilizes deep learning text classification techniques, fully integrates the characteristics of requirement text defects, and designs two defect detection classification models, TextCNN and BERT, for comparative training and performance evaluation based on the construction of a defective text dataset, and constructs a deep learning neural network based on BERT suitable for requirement text defect detection. Experimental results show that the method achieves more than 90% accuracy in defect detection, effectively makes up for the shortcomings of traditional manual review, and significantly improves the efficiency of software development and the quality of requirement text.**

***Index Terms*—requirements engineering, deep learning, defect detection**

I. Introduction

The requirements analysis phase assumes a pivotal role within the context of the software development life cycle, where the efficacy of a software product frequently hinges upon the quality of this developmental phase. The requirements analysis phase encompasses a sequence of pivotal stages, including requirements gathering, delineation, specification, authentication, and substantiation. Its terminal outcome, the requirements text, serves as a conduit for delineating the envisaged external comportment and attributes of the software system, thereby furnishing guidance and standardization to the software development trajectory. The quality of the requirements text exerts an indelible influence upon the assurance of software quality, thereby necessitating attributes encompass comprehensiveness and transparency, ease of comprehensibility, verifiability, unity both within and without, feasibility, and traceability. Hence, detecting deficiencies and conducting quality appraisals about requirement texts assume a cardinal role in safeguarding the caliber of software products.

Presently, the prevailing convention in requirement texts involves using natural language. However, the inherent dualistic nature of natural language introduces the potential for ambiguity within requirement texts. Furthermore, the diversity of domain knowledge among disparate audiences of requirement texts engenders a specialized lexicon that necessitates domain-specific acumen for comprehension. Furthermore, the intrinsic subjectivity associated with writers can occasion presentation irregularities within requirement texts. Consequently, a plethora of disparate defects often pervades requirement texts. Many investigations have demonstrated that defects inherent in requirement texts can precipitate profound and financially burdensome repercussions in software product development. Additionally, the temporal juncture at which a requirement problem is unearthed is inextricably linked to the expenses incurred during its rectification, thus underscoring the importance of early identification and rectification of requirement-related concerns for stakeholders invested in the project.

During the practical execution of the requirements analysis phase, the conventional practice entails utilizing manual reading techniques for requirements text inspection and review. However, these methodologies are characterized by their time-intensive nature and inherent issues. Moreover, these techniques impose considerable technical prerequisites on reviewers, augmenting the risk and onus associated with the text composition and review procedure. Consequently, pursuing a strategy for achieving efficient and precise detection of defects within requirements text has emerged as a focal point of scholarly investigation.

Given the robust advancement of neural networks and their commendable performance, the investigation endeavors to incorporate neural network technology into identifying defective textual content in requirements. This is achieved by constructing a classification model tailored for defective text, thus further substantiating the implementation of an automated system for detecting defects in requirement text. When addressing the task of recognizing defective text, attaining a heightened level of accuracy and comprehensive inclusivity not only alleviates the workload of requirement reviewers and enhances the review process's efficacy but also diminishes the technical threshold associated with defect detection. The facilitation empowers text authors to conduct rapid pre-checks of content during the composition phase. Such endeavors carry profound significance in minimizing requirement defects and augmenting the quality of requirement texts.

II. Ralated Work

In recent years, various methodologies have emerged to tackle the challenge of detecting defects within requirement texts. The conventional manual reading and inspection technique [1] has been criticized for its inefficiency and substantial consumption of human and material resources. In response, Nancy et al. [2] proposed a semi-automated defect detection method, which economizes manual reading time by identifying defects through predefined rules. However, the limited-intelligence matching approach primarily addresses the normative facets of requirement texts and cannot discern expressive defects inherent in the linguistic construction itself. Consequently, artificial intelligence has gained prominence in defect detection and quality analysis of requirement texts. Jani [3] proposed the Case-Based Reasoning (CBR) method, centered on constructing a requirement defect database, to identify defects by establishing similarity between new and database texts. Subsequently, Jani et al. [4] incorporated artificial neural networks into the CBR framework, enhancing its adaptive learning capabilities for case similarity assessment. Chantree et al. [5] harnessed machine learning to detect ambiguity defects, employing a 10-fold cross-validation approach across various heuristics to identify a significant subset of texts containing paired ambiguity defects. Additionally, Ferrari et al. [6] adopted a collective intelligence strategy, leveraging a model akin to a conceptual relationship graph, to uncover ambiguity-type defects within linguistic context. Rosadini et al. [7] underscored the potential of Natural Language Processing techniques in prioritizing manual analysis of shortcomings, alongside serving as a complementary tool for secondary validation post-manual review completion.

Text classification refers to the process of automatically categorizing the text in a document collection, and the rules required for classification can be derived from predefined or automatically learned by the machine through training. With the development of deep learning technology, text classification based on deep learning has gradually matured. Still, it has not been widely used in requirement text defect detection. Therefore, combining deep learning and requirement text defect detection has a wide range of research value and application potential.

This paper aims to analyze and think deeply about the defect detection techniques in requirement text and combine deep learning text classification techniques with requirement text defect detection. The research starts with constructing and preprocessing the requirement text defect dataset and designing diverse defect detection classification models for training and performance evaluation to realize an efficient defect detection function.

III. Method

A. Dataset Construction

1) Characterization and Classification of Requirements: Due to the lack of publicly available defective datasets for requirement texts, we constructed a defective dataset containing 10049 pieces of data after considering the characteristics of requirement texts to ensure balanced and stable data.

The requirements text has the following characteristics:

- Apparent brevity of information, with requirement statements generally limited to a concise span of 50 words or less.
- A large number of extraneous elements, including redundant delimiters, paragraph markers, and section numbering.
- Requirements text consists mainly of complex explanations related to business premises, functional specifications, and other complex issues. It is characterized by a high degree of specialization and limited scope and mainly serves the field of software development.

Based on thorough professional research and investigation, we provide the following definitions for fuzzy requirements, unverifiable requirements, and incomplete requirements:

- Fuzzy requirements: Requirements that require domain-specific specialized vocabulary are used, or requirements that do not use explicitly specialized terms. For example: "The server should have a stable bandwidth."
- Unverifiable requirements: Requirements that are not humanly controllable, cannot be verified, or are extremely costly to verify. For example: "The server should be able to run continuously for 1000 millennia."
- Incomplete requirements: Requirements that are incompletely expressed or have unplanned content. For example: "The specific return information of the interface has not been designed yet."

Consequently, we carefully curated a dataset comprising a total of 10,049 samples extracted from 215 requirement texts spanning diverse fields. Most of the data collected comes from requirement texts generated by experts and professional developers during the actual software development process, with their consent and privacy policy. The dataset encompasses a range of requirement types, including correct requirements, fuzzy requirements, unverifiable requirements, and incomplete requirements. To facilitate effective model training, validation, and testing, we partitioned the corpus into training, validation, and testing datasets, adhering to a ratio of 7:1:2.

2) Data Pre-processing: Preprocessing the dataset is necessary before proceeding with formal text classification model processing, which helps to avoid duplicating the work of processing the dataset and using the preprocessed dataset directly during training and parameterization. Here are the specific preprocessing steps:

- Segmentation: Given that the dataset size is not very large, this study employs a word-level-based segmentation approach rather than the more fine-grained word-based processing. This move introduced a priori knowledge in model training by performing word-based disambiguation. We used the Jieba lexer tool to perform lexer processing on all requirement texts and imported predefined keywords for the software requirement domain into the lexer dictionary through Jieba's interface.

- Removal of extraneous characters: The raw, unprocessed requirements text may contain subsection numbers, meaningless separators, or even garbled code in non-UTF-8 formats. These characters do not contribute to text categorization and should not usually be included in the lexicon. Therefore, while keeping the word order in the lexicon results unchanged, we removed separators and gibberish irrelevant to classification in each sample data. Given that requirement texts are usually short and the presence of stop words may change the meaning of an utterance, we did not further remove stop words to retain as much information as possible from the original requirement text.
- Constructing the lexicon: To map each word in the sample data to a word vector, we made a lexicon based on the segmentation results to provide a unique numeric ID representation for each word. In addition, to avoid affecting the IDs of the original words, we use an expanded dictionary to add new words.
- Uniform sample length: Most models can only accept fixed-size inputs, so we standardize the sample length. Specifically, we limit the number of words in a single sample, and if the number of words in a sample is less than the specified length, we use a particular filler ID to fill it. If the number of words in a sample is greater than the fixed length, we remove the top words. We limit the number of words per sample to 250 words, roughly three times the number of words in the longest statement in the existing sample. This is designed to avoid, as much as possible, deleting words to standardize the length.

3) Data Enhancements: Considering the limited scale of the dataset and suboptimal training outcomes, we employ data augmentation techniques to amplify the original dataset, aiming to enhance the quality of training. This study adopts the EDA text data enhancement algorithm, and the probability of random insertion and deletion is set to 0.04. Subsequently, the existing statements in the dataset are traversed and combined with the Chinese deactivation word list according to the distribution of non-deactivated words in statements. The EDA text data enhancement operation is executed according to the following rules:

1) Select non-deactivated words from the utterances for synonym replacement using the synonym table.
2) Randomly select non-discontinued words in the statement, extract and replace them with synonyms, and insert them into random positions in the statement, and this operation is repeated several times.
3) Two words are randomly selected, and their positions are exchanged. Again, this step is repeated several times.
4) Randomly removing a word from the utterance based on a predefined random deletion probability.

In addition to the EDA data enhancement method, this study employs the back-translation method for further data enhancement of the original dataset, i.e., translating the textual data into other languages and then translating it back to the original language to obtain a new corpus with the same labels. It should be noted that since the overall effect of back-translation is difficult to control fully, the data results generated by back-translation need to be further filtered:

- For requirement text utterances of shorter length contain more limited information. After back-translation, the enhancement effect may not be obvious, and the newly generated data may be very similar to the original text. Therefore, the new data generated by back-translation needs to be compared with the original text data to exclude those results that are too similar or identical.
- For the more informative requirement text, its expression may change significantly after back-translation, affecting its original labeling value. Therefore, the new data generated by back-translation needs to be compared with the text length of the original data to exclude those translation results whose size is too short or too long.

Through the above method, we have realized the enhancement of the demanded text data to ensure the data quality, which provides a rich corpus for the research.

B. TextCNN Classification Model

For the requirement text defect detection task, we adopt the model of TextCNN based on the Keras framework for the overall construction. To select the fitting word vector dimension judiciously, the study preconfigures three distinct dimensions—namely, 32, 64, and 128—after finalizing the optimal dimensionality by evaluating loss function performance across varying sizes.

Illustrated in Fig.1, a discernible trend emerges with the augmentation of training iterations, as evidenced by the progressive reduction of the loss function on the training set. Notably, when employing 32-dimensional word vectors, the loss converges and plateaus after a discernible descent, implying the insufficiency of feature expressiveness within vectors of the dimensionality, thus leading to an underfitting phenomenon. In contradistinction, the outcomes corresponding to 64- and 128-dimensional word vectors exhibit markedly superior performance compared to their 32-dimensional counterpart.

Moreover, Fig.2 delineates the loss trajectory within the validation set. Notably, when the dimensionality of 128-dimensional word vectors is adopted, the loss exhibits a diminishing trend before encountering a point where a contrary ascent is observed. The observation to some extent signifies the manifestation of overfitting within the model. Consequently, a judicious selection of 64-dimensional word vectors ensues.

Similarly, we apply an analogous methodology to ascertain the dimensions of the three convolutional kernels, configured as 3 * 64, 4 * 64, and 5 * 64. In concurrence, 64 convolutional

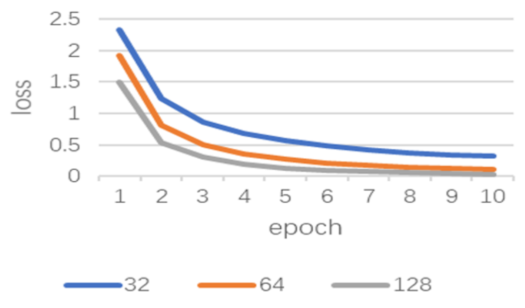

Fig. 1. Variation of Loss Values for Each Dimension of Word Vectors on Training Set

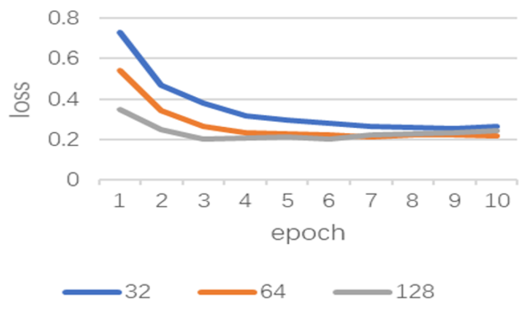

Fig. 2. Variation of Loss Values for Each Dimension of Word Vectors on Validation Set

kernels are crafted for each kernel size, yielding an output dimensionality of 64. Subsequently, the construction of the TextCNN model attains fruition.

C. Bert Classification Model

At the same time, we use the pre-trained Chinese BERT model to construct a text classification model. And we adopt the Keras_BERT framework to develop the classification model. The specific steps are as follows:

1) Firstly, we perform text preprocessing on the dataset, which includes filtering, filling, or cropping the text to ensure that each input text is the same length.
2) Next, we employ the BERT-wwm-ext version of the configuration file, dictionary file, and model file.
3) We generate two corresponding sequences for the input layer of BERT by modifying the Tokenizer. This involves splitting the text statement into its split components and their corresponding ordinal numbers while utilizing the [unused] tag to indicate character padding items.
4) Subsequently, we construct the model with BERT as the input layer. We extract the vector values corresponding to the output [CLS] and feed them into the Softmax layer for classification.
5) For optimization, we use the Adam optimizer to prevent overfitting. The loss function is set to categorical_crossentropy, and the learning rate is initially set to a small value of Adam(1e-5).

Upon constructing the model, it becomes crucial to determine two key parameters: tokensize and batchsize. We plan to conduct a comparative experiment to determine the appropriate values for these parameters.

To ensure the comprehensive preservation of utterance information, it is desirable to set tokensize in a way that encompasses the length of the utterance. However, excessively long sequences may dilute specific feature points and result in inefficient computation. Therefore, we analyzed the training results statistically using various sequence lengths to determine the optimal tokensize. The findings of this analysis are presented in Fig. 3.

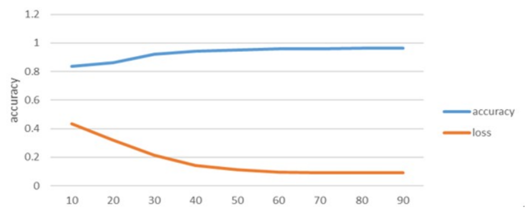

Fig. 3. Accuracy and Loss under Different Sequence Lengths

The presented figure demonstrates that as the sequence length increases, the model's accuracy shows a significant improvement when the sequence length remains below 70. Concurrently, there is a noticeable reduction in the loss value. However, beyond a sequence length of 70, additional increases in sequence length have minimal impact on the overall performance. Based on our analysis, we have determined that a sequence length of 70 effectively preserves the intrinsic feature information of the text. Consequently, for this paper, we have selected a tokensize of 70 as the final choice. Similarly, the batchsize of 16 has been determined through the same experimental design.

IV. Experiment

After completing the construction of TextCNN and BERT classification models, we perform the relevant training process for both on the same dataset and obtain the following TABLE I.

TABLE I
COMPARISON OF TEXTCNN AND BERT TRAINING DATA

Model	Accuracy	Training Period (epoch)	Training Time (h)
TextCNN	0.935	10	0.3
BERT	0.964	4	1

The analysis of the provided table readily reveals that the TextCNN model exhibits a swifter training pace. However, its corresponding accuracy rate falls short when juxtaposed against the BERT-based text classification model. The latter, though requiring an extended training duration, demonstrates expedited convergence and attains an elevated accuracy level. To glean a more comprehensive insight, We then conducted a comparison of the requirement accuracy data for each category as shown in TABLE II.

TABLE II
PRECISION DISPLAY OF VARIOUS DEFECT TYPES AT BERT AND TEXTCNN

Model Name	**Accuracy**				
	Correct	***Fuzzy***	***Unverifiable***	***Incomplete***	***Average***
TextCNN	0.957	0.897	0.986	0.876	0.929
BERT	0.957	0.952	0.998	0.959	0.966

For the detection of defective text, we prefer the recall rate to the accuracy rate, so we calculated the F1 metric of the two models by considering the recall rate and classification accuracy, and the specific data are shown in the TABLE III.

TABLE III
TEXTCNN AND BERT RECALL RATE AND F1 VALUE DISPLAY

Model	**Micro Average Recall Rate**	**F1 Metrics**
TextCNN	0.934	0.935
BERT	0.963	0.964

Combining the above data, it can be seen that BERT text classification has some lead in each comparison index, especially the accuracy of incomplete requirements has been dramatically improved, so we finally choose BERT as a text classification model.

After establishing our requirement text defect detection model leveraging BERT, we performed an intricate analysis of the temporal requisites associated with diverse text lengths in requirement detection. In parallel, we conducted comprehensive computations encompassing the entire temporal trajectory, from text reception to the subsequent provision of classification outcomes. This evaluated the mean temporal investment per individual text across varying text quantities and lengths. Detailed insights into these computations are meticulously presented in TABLE IV for reference and analysis.

TABLE IV
TEXT PROCESSING PERFORMANCE EVALUATION

Text	**Text Case 1**	**Text Case 2**	**Text Case 3**
Number	10	10	20
Length	30	50	50
Processing Time(ms)	0.0209	0.0299	0.0298
Calculating Results(s)	0.037	0.044	0.047

From the above table, it can be seen that as the length of the text increases, the time required for processing the text and calculating the results also increases, with an average of about 0.03-0.05 seconds required to calculate the final result per text.

V. CONCLUSION

In this paper, we constructed a defective text classification model using natural language processing and deep learning techniques. We realized the automatic detection function for three types of defective texts: fuzzy requirements, unverifiable requirements, and incomplete requirements. Our empirical investigations reveal that the classification model we propose attains an accuracy rate exceeding 90%. The performance allows for efficient and precise identification of issues and their corresponding typologies, thereby facilitating real-time quality assessment of requirement texts. Our approach substantially enhances the efficiency of software development endeavors and bolsters the quality of text editing. In the forthcoming trajectory of our research, we intend to expand the scope of the requirement text dataset and explore avenues for the complete automation of requirement text processing.

REFERENCES

[1] A A. Alshazly, and A M. Elfatatry, and M S. Abougabal Detecting defects in software requirements specification[J]. Alexandria Engineering Journal, 2014, 53(3): pp. 513-527.

[2] N. Vaish and A. Sharma, "Semi-Automated System Based Defect Detection in Software Requirements Specification document," in 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering. IEEE, 2018.

[3] H M. Jani, "Applying Case-Based Reasoning to software requirements specifications quality analysis system," in the 2nd International Conference on Software Engineering and Data Mining. IEEE, 2010, pp. 140-144.

[4] H M. Jani, and A B M T. Islam, "A framework of software requirements quality analysis system using case-based reasoning and Neural Network," in 2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining. IEEE, 2012, pp. 152-157.

[5] F. Chantree, B. Nuseibeh, A. De Roeck et al. Identifying nocuous ambiguities in natural language requirements[C]" in 14th IEEE International Requirements Engineering Conference (RE'06). IEEE, 2006: 59-68.

[6] A. Ferrari, and S. Gnesi, "Using collective intelligence to detect pragmatic ambiguities," in 2012 20th IEEE International Requirements Engineering Conference. IEEE, 2012, pp. 191-200.

[7] B. Rosadini, A. Ferrari, Gori G, et al."Using NLP to detect requirements defects: An industrial experience in the railway domain[C]" in International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, Cham, 2017: 344-360.

[8] I L. Margarido, and J P. Faria, and R M. Vidal et al. "Classification of defect types in requirements specifications: Literature review, proposal and assessment[C]" in 6th Iberian Conference on Information Systems and Technologies (CISTI 2011). IEEE, 2011: 1-6.

[9] A. Davis, S. Overmyer, K. Jordan, et al. "Identifying and measuring quality in a software requirements specification[C]" in 1993 Proceedings First International Software Metrics Symposium. IEEE, 1993: 141-152.

[10] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014.

[11] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position[J]. Biological cybernetics, 1980, 36(4): 193-202.

[12] A. Krizhevsky, I. Sutskever, G E. Hinton. "Imagenet classification with deep convolutional neural networks[C]" Advances in neural information processing systems. 2012: 1097-1105.

[13] S T. Hsu, C.Moon, P. Jones et al." A hybrid CNN-RNN alignment model for phrase-aware sentence classification[C]" in Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers. 2017: 443-449.

[14] J. Devlin, M W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2018.

[15] T. Lei, R. Barzilay , and T.Jaakkola, Molding CNNs for text: non-linear, non-consecutive convolutions[J].Indiana University Mathematics Journal, 2015, 58(3):págs. 1151-1186.DOI:10.18653/v1/D15-1180.