Jurnal Mantik, 7 (3) (2023), ISSN 2685-4236 (Online)

Published by:Institute of Computer Science (IOCS)

Jurnal Mantik

Journal homepage: www.iocscience.org/ejournal/index.php/mantik

Investigating deep learning approach for automated software requirement classification

Gregorius Airlangga
Department of Information System, Faculty of Engineering, Atma Jaya Catholic University of Indonesia, Jakarta, Indonesia

A R T I C L E I N F O

Article history:

Received Nov 9, 2023
Revised Nov 20, 2023
Accepted Nov 11, 2023

Keywords:

CNN;
Deep Learning;
Natural Language Processing;
Software Requirements;
Text Classification.

ABSTRACT

Requirement engineering has a critical role in software development. Various techniques have been invented to improve requirements quality. Recently, natural language processing (NLP) has drawn attention to the software engineering community, due to its capacity for automated requirements elicitation which accelerates software development. However, NLP through machine learning approaches has been discovered to be ineffective, caused by the variability in natural language, unbalanced dataset, heavy preprocessing tasks, and issues in hyperparameter tuning. A deep learning approach on the other hand has been found to handle enormous dataset without sophisticated preprocessing task and to improve classification quality for unstructured data such as images and text. This paper investigates how automated software requirements classification can be improved and how well deep learning approaches work. We aim to contribute an approach to minimize preprocessing requirements to improve classification accuracy from OpenScience tera-PROMISE repository. We also compare our approach with several techniques that have been previously tested. We find that our technique may improve the performance of an existing classification method. Finally, we present significant differences in the performance of approaches, such as for the sub-classification of nonfunctional requirements.

Corresponding Author:

Gregorius Airlangga,
Departement of Information System, Faculty of Engineering,
Atma Jaya Catholic University of Indonesia
Jl. Jenderal Sudirman No. 51, Jakarta, 10220, Indonesia.
Email: gregorius.airlangga@atmajaya.ac.id

1. INTRODUCTION

Software development has several processes that have to be achieved in order to build good quality software. Based on the waterfall point of view, the phases are comprised of five stages that started from analysis, design, coding, testing and culminated into maintenance phase (Stoudt, Vásquez, & Martinez, 2021), (Yathiraju, 2022), (Hadj-Mabrouk, 2020). Requirement engineering is the main aspect in analytic process that plays a critical role for the next development stages. Within the software requirement analysis, requirement engineering is the activity in which software stakeholder needs are understood (Laplante & Kassab, 2022), (Melegati, Goldman, Kon, & Wang, 2019), (Vogelsang & Borg, 2019). From

the activity, analyst attempt to identify the software purpose from the stakeholders and other factors that may give impact for the product quality.

In order to acquire software requirement such as functional and nonfunctional requirement, existing methods for the requirement gathering activity involve heavy interactions with the stakeholders, for example through meeting (Al-Saqqa, Sawalha, & AbdelNabi, 2020), (Lwakatare et al., 2019), interviews (Shastri, Hoda, & Amor, 2021), (Mohapatra & Rath, 2020), (Tam, da Costa Moura, Oliveira, & Varajão, 2020), focus group discussion (Kipper et al., 2021), (Day, Miles, Ginsburg, & Melvin, 2020), (Fatima et al., 2019) and brainstorming sessions (Pinto, Silva, & Valentim, 2020), (Braz et al., 2019), (Zorzetti et al., 2022). Those methods are easy to implement for the small software projects with small stakeholders quantity (Dwitama & Rusli, 2020), (Kasauli et al., 2021), (Seroka-Stolka & Fijorek, 2020), (Welikala et al., 2020).

However, the methods fail to scale to big projects with hundreds of thousands of stakeholders that also involved thousands project documents. In addition, gathering all possible requirements from the early beginning is difficult and mostly the requirements are often discovered in an ad hoc fashion relatively late in the development process. Based on the problems, the automatic requirement gathering approach that could facilitate early detection mechanism may useful since it enable system level constraints to be considered and incorporated into early architectural designs as opposed to being refactored at a later time (Martinez et al., 2020), (Khraisat & Alazab, 2021), (Amershi et al., 2019).

It also may enable to quickly analyze large and complex project in order to search for software requirements. Therefore the technology could be used for supporting a software analyst in order to minimize the error-prone task of manually discovering software requirement. Recently, natural language processing (NLP) has drawn attention to the software engineering community, due to its capacity for automated requirements elicitation which accelerates software development process. Many works have been conducted by enormous researchers in NLP and software engineering domain that attempt to use machine learning approaches (Giray, 2021), (Watson et al., 2022), (Adnan & Akbar, 2019).

However, those approaches have been discovered to be ineffective, caused by the variability in natural language, unbalanced dataset, heavy preprocessing task and issues in hyperparameter tuning. In contrast, a deep learning approach has been found to handle enormous dataset without sophisticated preprocessing task and to improve classification quality for unstructured data such as images, speech signals and texts (Chatterjee et al., 2021). This paper investigates how automated software requirement classification can be improved using deep learning model. In this work we investigate several deep learning models with two preprocessing mechanism that commonly used in text classification problem. We also aim to contribute to minimize preprocessing requirements approach to improve classification accuracy from OpenScience tera-PROMISE repository. This work is comprised by five sections. Section 1 is introduction, Section 2 is related work, Section 3 is Experiment, Section 4 is discussion and the last is conclusion.

Automated software requirement classification has been investigated by several researchers. In (Wang, Teoh, Lu, & Choi, 2020) build Open-Science tera-PROMISE repository that contained the software requirements which are composed of 11 nonfunctional requirements such as availability, legal, look and feel, maintainability, operational, performance, scalability, security and usability. In addition they also add functional requirements which have 40.9% percentage from all dataset gathering from 15 master student software projects. Furthermore, they propose indicator terms mining to classify 9 non-functional software requirements. They propose preprocessing method to extract indicator keyword term “mined” from the training set. After that, the keyword term would be used as an input into weighted indicator term for classifier. As evaluation, they used Leave-one-out cross validation with several metrics measurements such as

precision, recall, and specificity. Another experiment was conducted by (Armouty & Tedmori, 2019).

They used several preprocessing method such as POS tagging, Entity tagging, Temporal tagging, co-occurrence and Regular expression rule that tried to normalize the data and accentuate the keyword weight in each sentence. After that, they used C4.5 decision tress classifier for classifying NFR and FR. As an evaluation they used 10 fold cross validation. They used precision, recall and F1 as their metrics measurement. In addition, they tested the proposed preprocessing approach to classify 9 attributes from Open Promise dataset. They found that Naïve Bayes classifier with their preprocessing method give the best result, the score is around 90-91% for average precision, recall and F1. Another experiment also was conducted by (Rachman, 2020), in this experiment, they propose preprocessing method to extract word features using automatic selection of k best feature which k is the number of extracted keyword. As a classifier, they used Support Vector Machine for classifying 2 classes of NFR and FR.

In addition they also propose support vector machine for classifying 4 nonfunctional software requirements. They choose NFR attributes collections that have percentage quantity above 8%. As an evaluation, they used motecarlo cross validation method. As an evaluation matrics they also used precision, recall and F1. According to their experiment, classification results yield best results for classifying FR and NFR for word features without automatic feature selection and the condition is also prevailed with their 4 classification problem. To the best of our knowledge, all of previous approaches have many preprocessing task and time consuming, also the conducted evaluation test is not genuinely build a guarantee to demonstrate prediction result based on the data that has not been seen before. In this paper, we contribute to minimize the preprocessing and feature selection tasks using convolutional neural network (CNN) with word2vec approach. As a validation, we also use the data that never seen before by the model in binary classification and multi classification problem.

2. METHODS

In this section, preprocessing techniques using TfIDF and Word2Vec are explained. In addition, we describe convolutional neural network model.

2.1. Term Frequency (TF)

In large documents, one of the most successful and widely used as feature extraction is Term Weighting TF-IDF (Jang, Kim, & Kim, 2019). TF-IDF is the combination of two methods: Term Frequency (TF) and Inverse Document Frequency (IDF). The main aspect to be considered in finding information from heterogeneous document collections are weighting terms. Terms can be considered as a form of words, phrases or other results of indexing units in a document that can be used to determine the document context.

After terms type is already determined, each word would be treated as an indicator, namely the term weight. TF (Term Frequency) is the appearance term frequency in the document. If the number of occurrences of a term (high TF) in the document is greater, then the value of conformity would be greater too. In Term Frequency (TF), there are three TF methods that are commonly used. First is Binary TF (binary TF), the method is only pay attention to whether or not a word or term exists in the document, if the word exists in the document, then the TF value is one (1), otherwise it would give a zero value.

Second is Pure TF (raw TF), the method is the one of the TF variants that only gives the TF weight based on the number of term occurrences in the document. For example, if the word appears ten (10) then the word will be worth as ten (10). The third method is Logarithmic TF which tries to avoid the dominance of documents that contain fewer terms in the query but have a high frequency. Logarithmic TF is expression is described in first equation.

$$TF = 1 + log\ (wt, d),\ (wt, d) > 0 \quad (1)$$

$$TF = 0,\ (wt, d) = 0 \quad (2)$$

Where (wt, d) is the term (t) frequency in the document (d). So, if a word or term appears in a document for 7 times then weights value would be 1 + log (7) = 1.845. However, if the term is not contained in the document, then the weight value is zero (0). Fourth is TF normalization that uses a comparison between the term frequency with the maximum value of the whole or a collection of terms frequency that contained in a document. TF normalization is commonly expressed as a second equation.

$$TF = 0.5 + 0.5x\left[(wt, d)/\left(\max(wt', d: t', d\varepsilon d)\right)\right] \quad (3)$$

2.2. Inverse Document Frequency (IDF)

IDF (Inverse Document Frequency) is a calculation method of how the terms are widely distributed in the collection of documents. IDF shows the relationship of availability of a term in all documents. If the number of documents that contain terms in document is fewer, then the IDF value would be greater. IDF is calculated using the second equation.

$$IDF_j = log\ \frac{D}{df_j} \quad (4)$$

Where D is the number of all documents in the collection and dfj is the number of documents containing the term (wj).

2.3. Term Frequency Inverse Document Frequency (TFIDF)

If we combine the pure TF concept and IDF thus the general formula for TFIDF could be generated by calculating the raw TF with IDF formula by multiplying TF values with IDF values as expressed in the fourth equation.

$$W_{ij} = tf_{ij}\ idf_j \quad (5)$$

In the fifth equation, Wij is the term weight for the document (dj) and tij is the number of term (t) occurrences of the term (ti) in the document (di). D is the number of all documents in the database and d/j is the number of documents that containing the term (tj). Regardless of the value of $TYij$. if $D = dfj$, then the result will be 0 (zero), due to the result of log 1. Therefore, we can add a value of 1 on the IDF^i sides, so that the calculation of the weight becomes as expressed in sixth equation.

$$W_{ij} = tf_{ij}\ log\ \frac{D}{df_j} + 1 \quad (6)$$

2.4. WORD2VEC

Word2Vee is a vector representation of words that are made by Google. Word2Vec can represent the words meaning, we can measure several vectors as a comparison. If we measure the distance between the words "Taiwan" and "Taipei" and the word "Indonesia" with "Jakarta", it will be found that the distance will appear in the adjacent number since the two words are basically the names of the country and capital. Because the meaning is close together, the vector value will also be close together. The results of these relations become references in representing a word into a vector. By the structure, Word2 Vec has 2 models: continuous bow of word (CBOW) and skipgram.

a. Continous Skip Gram: This is a model that introduced by (Ghalyan, 2020) Architecturally, Skip-Gram uses the current word (as input) to predict the context (as a target) around it. Skip-Gram will study the probability distribution of words in context with windows that has been specified. For example, the context is "the grass is greener where you water it" with the value of word windows = 2. To represent the context in Skip-Gram architecture, we must convert each word to one-hot encoded vectors, a forward-backward training Skip-Gram illustration with a random value on the weight (W and W*), with $w(t)$ = "greener" as input, and $w(t-2)$ = "grass", w $(t-1)$ = "is", w

$(t+1)$ = "where", w $(t+2)$ = "you" as the target. After the output is obtained, it is necessary to minimize the error value using, cross entropy method. After calculating the error value, backprogation mechanism will be conducted, by calculating the loss function gradient of all existing parameters using partial derivative equation. In the backprogation, a parameter update process occurs, the update mechanism parpose is to reduce of add the old weight (W) with the obtained gradient value (Gradient Descent) until the minimum errof value is reach convergence condition.

b. Continous Bag of Words (BOW): This is a model that predicts the current word (as a target) from the context (as input) which are located around the current word (Alzubaidi et al., 2021). It could be said that CBOW is the opposite way of Skip-Gram, where CBOW will also study the prohability distribution of context with windows that has been specified. The context is also uses similar onehot encoded vectors as Skip-Gram. CBOW forward-backward training mechanism with random values on weight (W), with $w(t-2)$ = "grass", w $(t-1)$ = "is", w $(t+1)$ = "where ", $W(t+2)$ = " you "as input, and $w(t)$ = "$grcener$" as the target. After the output probability is obtained, then the next step is similar to the Skip-Gram mechanism.

2.5. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is one of the most advanoed machine learning methods which is usually implemented to process two-dimensional data. Since CNN is one of the Deep Learning Neural Network variants, it has a network level using feed forward and back propagation mechanism in order adjust optimal weight (Narkhede, Bartakke, & Sutaone, 2022) and (Liu et al., 2023).

In 2-dimensional array, if CNN is assigned to the features of the human face images, then the first layer reflects strokes in different directions, on the second layer features is containing richer features such as the shape of the eyes, nose, and mouth that begin to appear. Convolution calculation plays a key role in the feature extraction in the first layer, then the effect Pooling/Combining on the third layer will form a combination of eye, nose and mouth features that will be concluded with the face of a particular person. Similar with the Neural Network in general, CNN has several hidden layers (hidden layers) from an input in the form of a single vector. In each hidden layer, there are several neuroes that are used for feature mapping. Each neuron is connected from the beginning until the last layer which finally represents the final elass classification.

Experiment

Dataset

The dataset used in this study is OpenScience Tera-Promise repository. It contains twelve software requirement categories including eleven sets of nonfunctional requirements and one set of functional requirements. The data was generated by Jane Cleland-Huang research, from fifteen master student projects. The total number of data is 625 rows. One of the main challenges in the dataset is unbalanced elasses pereentage that could lead a negative result for the elassifiers. To resolve the problem, several rescarchers have attempted to conduct several preprocessing and feature extraction techniques. In this work we use TFIDF and word2vec. Another challenge of the dataset is a data collision that each sentence in the class category has the same vector location. The conditions add the difficulty level for the classifier to separate the sentence in correct group. To show the data visualization for analysis approach, we build a data visualization from the dataset. The data visualization is presented in figure 1. The visualization was conducted using three main steps: the raw data would be converted into vector using word to indices mechanism, after that we train the data using one layer LSTM method into twelve class categories to get the training weight. Finally, the dimension would be reduced using t-sne method into two dimensions for simplifying visualization into cartesian diagram. By using t-sne,

perplexity parameter is 12 and n_component is 2, t-sne method is trained in 500 iterations epoch.

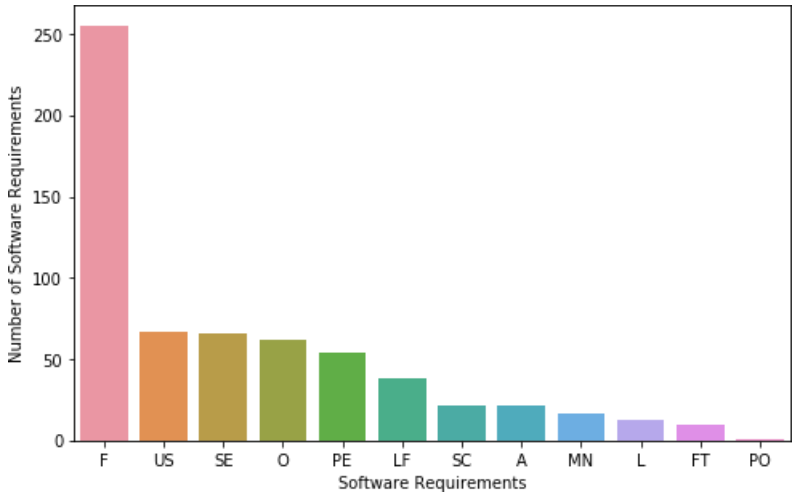

Figure 1. Software Requirements Graphic

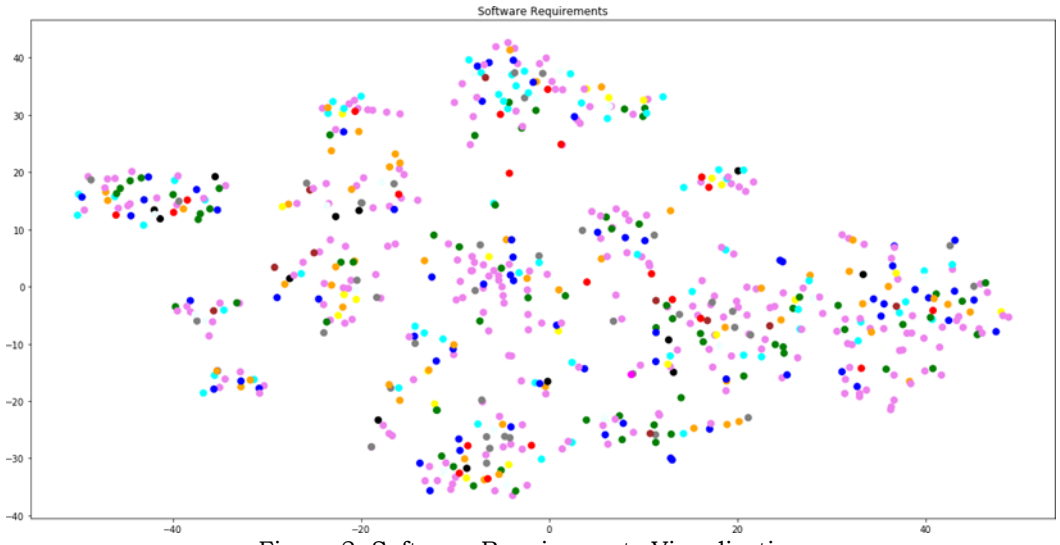

Figure 2. Software Requirements Visualization

From the figure 2, the x coordinates have interval between -60, -60 and y coordinates have interval between $-40, -40$. As we seen in figure 2, the nonfunctional requirements mostly collide with functional requirements (Violet color). From our observations, each functional requirement is in each interval. It means that the nonfunctional requirement tends to misclassify into functional requirement, in addition the classifier could tend to achieve overfitting situation. In this work, we classify the dataset into two categories. To minimize overfiting problem, we use minibatch gradient descent in our deep learning model, in addition we set a validation size in each epoch which containing, 20% of total datasets. As evaluation, we use two eategory of dataset to evaluate the accuracy. The first eategory is the data that divided into nonfunctional and functional requirement where all the nonfunctional requirement is concatenated. Socondly, we consider the dataset that have percentage more than 8% as presented at table 1 therefore

we just use 4 nonfunctional attributes such Usability, Security, Operational and Performance.

Table 1. Dataset Size

Quality Attributes	Number of Requirement	Percentage of Requirement
Funcionality (FT)	255	40.8
Usability (US)	67	10.72
Security (SE)	66	10.56
Operational (O)	62	9.92
Performance (PE)	54	8.64
Look and Feel (LF)	38	608
Scalability (SC)	21	3.36
Availability (A)	21	3.36
Maintainability (MN)	17	272
Legal (L)	13	2.08
Fault Tolerance (FT)	10	1.6
Portability (PO)	1	0.16
Tolal	625	100

Preprocessing

In general, text documents are going to use different forms of a word such as make, makes and making. Additionally, there are families of derivationally related words with similar meanings such as build, conduct, develop. In classification problem, the variation of text does not give a lot of impact for the accuracy. Therefore, stemming and lemmatization could be implemented in preprocessing phase to generalize the word that has similar meaning.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. In our experiment instead we use stemming that might be resulted into incomplete sentences. We use cleansing, regular expression and lemmatization that usually refers to doing things properly with the use of a vocabulary and morphological analysis of words. Those preprocessing methods would combined with TFIDF and word2vec only for non-Deep Learning Model for the comparison purpose.

Feature Extraction

Related works show variant feature extraction method, (Sueno, Gerardo, & Medina, 2020) used indicator selection method. By using the methods, it automatically chooses alternate keywords that were evaluated for selecting terms from the training set. Since we do a comparison study between deep learning and machine learning method, we choose two variants of feature extraction which are commonly used in text document: TFIDF, word2vec using googlenews300d. In order to reduce the preprocessing, we only used word2vec without preprocessing that used before word2vec embedding layer in our Deep Learning Model.

Classifiers & Deep Learning Variants

We use several strong classifiers variants instead weak classifiers since those methods learn the pattern to build an inference model to predict the test data and this mechanism is similar with Deep Learning Model. In this experiment we use logistic regression, SVM, Naïve Bayes. All classifiers are worked with preprocessing and feature extraction that described at section 4.3. In addition, Naïve Bayes and SVM are proved to have a good result that presented by another researcher using similar dataset in (Sueno et al., 2020) and (Rahat et al., 2019). We propose 9 variants of deep learning models: GRU, LSTM, Bidirectional LSTM, Bidirectional GRU, CNN, CNNGRU, GRUCNN, CNNLSTM, LSTMCNN. In this experiment, we propose our CNN architecture for software requirement classifications model. We also combine our approach with other neural network models

which are commonly used in text classification problem such as GRU, LSTM, Bidirectional GRU and Bidirectional LSTM.

Evaluation

We use train test split where 80% data would be generated as training dataset and 20% data would be generated as testing dataset. Evaluation metrics are comprised of 4 measurements which are accuracy, precision, recall and F1. We also present the 4 measurements from the training and test dataset. Training set is the previously seen data, in contrast test set is the unseen data. Test dataset is randomly generated using train_test_split from scikit-learn library.

Proposed CNN Architecture

Proposed CNN architecture is described on Table 2. First, we build an embedding layer to build a new matrix from the matrix of vectors that represent each word in the sentences. The embedding layer size is several sentences as row values and the columns are number of outputs which have 300 embedded vector size. After all embedding layers have been constructed then the three-layer convolution network will be assigned to do convolution task. Each convolution has the task to convolve 1 dimension of the embedded vector in 3 filter sizes. It means that the convolution neural network would extract the most effected feature that containing of 3 numbers of words in every sentence.

Maxpooling mechanism also will be conducted in each convolution process sequentially to extract the most valuable weight feature in the sentence. After three layers of convolution-maxpooling are calculated, then we concatenate all the result and flatten the output matrix to build a one vector matrix. After that, to handle overfitting problem we used single layer dropout mechanism with rate 0.4. The dropout layer will select active neurons that could participate in calculation process randomly. Lastly, all the matrices would be calculated into the last neuron layer that have dimension size with the number of data times number of classes. In this proposed CNN, the activation layer in output is used softmax and categorical loss entropy is initiated as loss function. This initialization is assigned for both binary and multiclassification problems.

3. RESULT AND DISCUSSION

3.1. NFR and FR Classification

NFR and FR classification results are presented at table 3. Interestingly, the other strong classifier: Naive Bayes, Logistic Regression and SVM are capable to classify the binary classification problem with higher result for all the evaluation measurement such as training and testing accuracy, F1, recall and precision. This statement is related with the (Jang, Kim, & Kim, 2019) that said the deep learning approach cannot do better accuracy in little quantity of data size. In contrast with other deep learning variant that give negative result, our CNN architecture could give a best result even comparing with another machine learning. approach. It means that our model could automatically extract the semantic and most important features in the sentences of the classes.

Table 2. Deep CNN Architecture

Layer Name	Parameter = Value	Output Shape	Total parameters
Input Layer	Shape = (Number of Data x 300)	(None, 300)	0
Embedding Layer	Shape = (Number of Data x 300 x 300)	(None, 300,300)	519600
Convolution 1D (a)	Kernel Size = 3 Padding = same Activation = Relu	(None, 300,64)	57664
Corvolution 1D (b)	Kernel Size = 3 Padding = same	(None, 300,64)	57664

Corvolution 1D (c)	Activation = Relu Kernel Size = 3 Padding = same Activation = Relu	(None, 300,64)	57664
Max Pooling 1D (a)	strides = 1	(None, 3, 64)	0
Max Pooling 1D (b)	strides = 1	(None 3,64)	0
Max Pooling 1D (c)	strides = 1	(Nive, 3,64)	0
Concatenate Layer		(None, 9, 64)	0
Flatten Layer		(None: 576)	0
Dropout Layer	Rate = 0.4	(None, 576)	0
Dense Layer	Nearon = 2 or 4	(None, 2)	1026

Table 3. NFR and FR Result

Model Name	Classifier	Training Accuracy	Testing Accuracy	Testing Precision	Testing Recall	Testing F1
Al	Naive Bayes	95.2	91.2	91	91	91
	SVM	99.2	92.8	93	93	93
	Logistic Regression	99.8	92.8	93	93	93
A2	Naive Bayes	99.8	81.6	82	82	82
	SVM	91.8	85.6	86	86	86
	Logistic Regression	81.6	76.8	77	77	77
DI	MLP	98.8	90.4	91	90	90
	GRU	99.8	90.4	91	90	91
	LSTM	99.8	90.4	91	90	90
	Bidirectional LSTM	99.8	88	88	89	88
	Bidirectional GRU	99.8	88	89	88	88
	CNN	99.8	87.2	87	87	87
	CNNGRU	99.6	91.2	92	91	91
	GRUCNN	99.8	88	88	88	88
	CNNLSTM	99.8	88	88	88	88
	LSTMCNN	99.6	85.6	86	86	85
D2	MLP	100	85.6	86	86	86
	GRU	99.8	88	88	88	88
	LSTM	99.8	88	88	88	88
	Bidirectional LSTM	99.8	92.8	93	93	93
	Bidirectional GRU	99.8	88	89	89	89
	CNN	99.8	94.4	95	94	94
	CNNGRU	99.6	90.4	91	90	90
	GRUCNN	100	88	89	89	89
	CNNLSTM	99.8	92.8	93	93	93
	LSTMCNN	97.6	83.2	86	83	83

3.2. NFR Attributes Classification (Usability, Security, Performance, Look and Feel)

For the multiclassification problem, the result is presented at table 4. All the methods, even for Nalve bayes, Logistic Regression, SVM and mostly other Deep Learning variants show lower training accuracy, testing accuracy, precision, recall and F1. It means that those methods fail to extract the features from the sentences in each category. Even for using word2vec compared with TFIDF. However, our CNN give very good training

accuracy, testing accuracy, precision, recall and F1 when we combine the method using word2vec as our word embedding vector. Interestingly, the combination method from bidirectional LSTM and CNN are given little worse result comparing pure CNN. From the confusion matrix, the CNN on training data perfectly classifies all the training data into each category. Meanwhile, in testing data the CNN only misclassified Usability into operational and performance and security is misclassified as performance. For the Security and Operational altributes. The CNN approach could effectively classify based on their classes.

Table 4. Four NFR attributes classification result

Model Name	Classifier	Training Accuracy	Testing Accuracy	Testing Precision	Testing Recall	Testing F1
B1	Naïve Bayes	99.49	80	83	80	80
	SVM	99.49	80	83	80	80
	Logistic Regression	100	80	81	80	80
B2	Naïve Bayes	86.93	76	76	76	76
	SVM	98.49	76	78	76	75
	Logistic Regression	100	78	80	78	77
	MLP	96.9	76	77	76	76
	GRU	99.49	74	77	76	75
	LSTM	100	76	77	84	80
	Bidirectional LSTM	100	84	85	86	85
	Bidirectional GRU	100	84	86	89	87
	CNN	100	74	75	90	82
	CNNGRU	99.49	80	80	85	83
	GRUCNN	100	76	80	80	79
	CNNLSTM	100	84	86	84	84
	LSTMCNN	100	70	71	78	73
E1	MLP	100	77.5	82	78	78
	GRU	100	81.6	82	82	82
	LSTM	88.5	75.51	79	76	76
	Bidirectional LSTM	100	86	87	86	86
	Bidirectional GRU	100	86	87	86	86
	CNN	100	93.8	95	94	94
	CNNGRU	100	77.5	80	80	80
	GRUCNN	98	85.71	88	86	86
	CNNLSTM	96.5	91.83	93	92	92
	LSTMCNN	100	79.59	81	80	80

4. CONCLUSION

In this research, we have embarked on an ambitious journey to redefine the classification of software requirements through the lens of deep learning, setting a new benchmark in this domain. Our model, meticulously compared against a range of machine learning techniques and existing research, stands out for its high training and testing accuracy, precision, recall, and F1 score. These performance indicators are not just numbers; they represent a significant leap in our ability to classify software requirements with minimal preprocessing, a task traditionally fraught with complexities. A core discovery of our work is the differential performance of the TFIDF method in traditional classifiers versus the

Word2Vec method in deep learning models. This distinction is crucial as it highlights the nuanced application of these methodologies in the context of software engineering. Our research contributes to the field by proposing a refined Word2Vec method, tailored to grasp the intricate semantic relations specific to software engineering, a noticeable departure from the broader scope of the generic Google Word2Vec model. This research serves as a bridge, connecting the current state of knowledge with the potential for future exploration. We have not only developed a model that effectively classifies software requirements but have also laid the groundwork for further advancements in this field. The specialized Word2Vec method we propose opens new avenues for exploration, suggesting that a more nuanced understanding of semantic relationships in software engineering can lead to greater accuracy in requirement classification. The implications of our findings are manifold. For practitioners in software engineering, our model offers a more precise tool for requirement classification, potentially reducing errors and enhancing efficiency in the development process. For researchers, the insights gleaned from our comparison of TFIDF and Word2Vec methods provide a foundation for further study into the optimization of text processing techniques in software requirement analysis. As we look to the future, it is evident that the journey of refining and enhancing software requirement classification models is far from complete. We advocate for continued research in developing more sophisticated versions of the specialized Word2Vec method, possibly integrating it with other advanced machine learning techniques to create more dynamic and adaptable models. Such future research should not only strive to improve the accuracy of classification but also explore the broader implications of these methods on the software development lifecycle, including requirement gathering, project management, and quality assurance. The potential applications of these advancements are vast, offering a promising horizon for both academic inquiry and practical application in software engineering. In conclusion, our study marks a pivotal step in the journey towards more sophisticated and nuanced software requirement classification. The path ahead is rich with opportunities for further exploration and innovation, promising to unlock new potentials in the realm of software engineering.

REFERENCES

Adnan, K., & Akbar, R. (2019). Limitations of information extraction methods and techniques for heterogeneous unstructured big data. International Journal of Engineering Business Management, 11, 1847979019890771.

Al-Saqqa, S., Sawalha, S., & AbdelNabi, H. (2020). Agile software development: Methodologies and trends. International Journal of Interactive Mobile Technologies, 14(11).

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., ... & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of big Data, 8, 1-74.

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., ... & Zimmermann, T. (2019, May). Software engineering for machine learning: A case study. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 291-300). IEEE.

Armouty, B., & Tedmori, S. (2019, April). Automated keyword extraction using support vector machine from Arabic news documents. In 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT) (pp. 342-346). IEEE.

Braz, R. D. S., Merlin, J. R., Freitas Guilhermino Trindade, D., Eduardo Ribeiro, C., Sgarbi, E. M., & Junior, F. D. S. (2019). Design thinking and scrum in software requirements elicitation: A case study. In Design, User Experience, and Usability. Design Philosophy and Theory: 8th International Conference, DUXU 2019, Held as Part of the 21st HCI International Conference, HCII 2019, Orlando, FL, USA, July 26–31, 2019, Proceedings, Part I 21 (pp. 179-194). Springer International Publishing.

Chatterjee, R., Ahmed, A., Anish, P. R., Suman, B., Lawhatre, P., & Ghaisas, S. (2021, September). A Pipeline for Automating Labeling to Prediction in Classification of NFRs. In 2021 IEEE 29th International Requirements Engineering Conference (RE) (pp. 323-323). IEEE.

Day, L. B., Miles, A., Ginsburg, S., & Melvin, L. (2020). Resident perceptions of assessment and feedback in competency-based medical education: a focus group study of one internal medicine residency program. Academic Medicine, 95(11), 1712-1717.
Dwitama, F., & Rusli, A. (2020). User stories collection via interactive chatbot to support requirements gathering. TELKOMNIKA (Telecommunication Computing Electronics and Control), 18(2), 890-898.
Fatima, R., Yasin, A., Liu, L., Wang, J., Afzal, W., & Yasin, A. (2019). Improving software requirements reasoning by novices: a story-based approach. IET Software, 13(6), 564-574.
Ghalyan, I. F. J. (2020). Estimation of ergodicity limits of bag-of-words modeling for guaranteed stochastic convergence. Pattern Recognition, 99, 107094.
Giray, G. (2021). A software engineering perspective on engineering machine learning systems: State of the art and challenges. Journal of Systems and Software, 180, 111031.
Hadj-Mabrouk, H. (2020). Application of Case-Based Reasoning to the safety assessment of critical software used in rail transport. Safety science, 131, 104928.
Jang, B., Kim, I., & Kim, J. W. (2019). Word2vec convolutional neural networks for classification of news articles and tweets. PloS one, 14(8), e0220976.
Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., & de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. Journal of Systems and Software, 172, 110851.
Khraisat, A., & Alazab, A. (2021). A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. Cybersecurity, 4, 1-27.
Kipper, L. M., Iepsen, S., Dal Forno, A. J., Frozza, R., Furstenau, L., Agnes, J., & Cossul, D. (2021). Scientific mapping to identify competencies required by industry 4.0. Technology in Society, 64, 101454.
Laplante, P. A., & Kassab, M. (2022). Requirements engineering for software and systems. Auerbach Publications.
Liu, X., Zhou, G., Kong, M., Yin, Z., Li, X., Yin, L., & Zheng, W. (2023). Developing multi-labelled corpus of twitter short texts: a semi-automatic method. Systems, 11(8), 390.
Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., ... & Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. Information and Software Technology, 114, 217-230.
Martinez, B., Reaser, J. K., Dehgan, A., Zamft, B., Baisch, D., McCormick, C., ... & Selbe, S. (2020). Technology innovation: advancing capacities for the early detection of and rapid response to invasive species. Biological Invasions, 22(1), 75-100.
Melegati, J., Goldman, A., Kon, F., & Wang, X. (2019). A model of requirements engineering in software startups. Information and software technology, 109, 92-107.
Mohapatra, H., & Rath, A. K. (2020). Fundamentals of software engineering: designed to provide an insight into the software engineering concepts. BPB Publications.
Narkhede, M. V., Bartakke, P. P., & Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. Artificial intelligence review, 55(1), 291-322.
Pinto, R., Silva, L., & Valentim, R. (2020, March). Managing sessions of creative requirements elicitation and assessment. In Proceedings of the 35th Annual ACM Symposium on Applied Computing (pp. 1355-1362).
Rachman, F. H. (2020, October). Twitter sentiment analysis of Covid-19 using term weighting TF-IDF and logistic regresion. In 2020 6th Information Technology International Seminar (ITIS) (pp. 238-242). IEEE.
Rahat, A. M., Kahir, A., & Masum, A. K. M. (2019, November). Comparison of Naive Bayes and SVM Algorithm based on sentiment analysis using review dataset. In 2019 8th International Conference System Modeling and Advancement in Research Trends (SMART) (pp. 266-270). IEEE.
Seroka-Stolka, O., & Fijorek, K. (2020). Enhancing corporate sustainable development: Proactive environmental strategy, stakeholder pressure and the moderating effect of firm size. Business Strategy and the Environment, 29(6), 2338-2354.
Shastri, Y., Hoda, R., & Amor, R. (2021). The role of the project manager in agile software development projects. Journal of Systems and Software, 173, 110871.
Stoudt, S., Vásquez, V. N., & Martinez, C. C. (2021). Principles for data analysis workflows. PLOS Computational Biology, 17(3), e1008770.
Sueno, H. T., Gerardo, B. D., & Medina, R. P. (2020). Multi-class document classification using

support vector machine (SVM) based on improved Naïve bayes vectorization technique. International Journal of Advanced Trends in Computer Science and Engineering, 9(3).

Tam, C., da Costa Moura, E. J., Oliveira, T., & Varajão, J. (2020). The factors influencing the success of on-going agile software development projects. International Journal of Project Management, 38(3), 165-176.

Vogelsang, A., & Borg, M. (2019, September). Requirements engineering for machine learning: Perspectives from data scientists. In 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW) (pp. 245-251). IEEE.

Wang, G., Teoh, J. Y. C., Lu, J., & Choi, K. S. (2020). Least squares support vector machines with fast leave-one-out AUC optimization on imbalanced prostate cancer data. International Journal of Machine Learning and Cybernetics, 11(8), 1909-1922.

Watson, C., Cooper, N., Palacio, D. N., Moran, K., & Poshyvanyk, D. (2022). A systematic literature review on the use of deep learning in software engineering research. ACM Transactions on Software Engineering and Methodology (TOSEM), 31(2), 1-58.

Welikala, R. A., Remagnino, P., Lim, J. H., Chan, C. S., Rajendran, S., Kallarakkal, T. G., ... & Barman, S. A. (2020). Automated detection and classification of oral lesions using deep learning for early detection of oral cancer. IEEE Access, 8, 132677-132693.

Yathiraju, N. (2022). Investigating the use of an Artificial Intelligence Model in an ERP Cloud-Based System. International Journal of Electrical, Electronics and Computers, 7(2), 1-26.

Zorzetti, M., Signoretti, I., Salerno, L., Marczak, S., & Bastos, R. (2022). Improving agile software development using user-centered design and lean startup. Information and Software Technology, 141, 106718.