

# Using RECURRENT NEURAL NETWORKS for CLASSIFICATION of Natural Language-based NON-FUNCTIONAL REQUIREMENTS

Rajesh Kumar Gnanasekaran, Suranjan Chakraborty, Josh Dehlinger and Lin Deng

*Department of Computer and Information Sciences, Towson University, Towson, MD 21252, USA*

## Abstract

In software projects, non-functional software requirements (NFRs) are critical because they specify system quality and constraints. As NFRs are in natural language, accurately analyzing NFRs requires domain knowledge, expertise, and significant human efforts. Automated approaches that can help identify and classify NFRs can lead to reduced ambiguity and misunderstanding among software engineers, decreasing developmental costs and increasing software quality. This paper investigates the effectiveness of leveraging machine learning techniques to automatically classify various types of NFRs. Specifically, we develop and train a recurrent neural network model, which has been evaluated to be effective in handling sequential natural language text, to classify natural language NFRs into five different categories: maintainability, operability, performance, security, and usability. We evaluate and detail insights from the experimental study performed on two data sets that contain almost 1,000 NFRs. The experimental results show that this approach can classify NFRs with a precision average of 84%, recall of 85%, F1-score near 84%, and classification accuracy of 88% on the testing data set. As indicated by the results, applying appropriate machine learning techniques can help reduce manual efforts, eliminate human mistakes, facilitate the software requirements analysis process, and lessen developmental costs.

## Keywords

Requirements Engineering, Machine Learning, Natural Language Processing

## 1. Introduction

Requirements Engineering (RE) is an essential phase in the software development process [1]. Software requirements are broadly classified into two main categories, functional requirements (FRs) and non-functional requirements (NFRs). NFRs define a software system's constraints and quality expectations concerning the underlying intricacies involved in integrating disparate subsystems and different architectural layers. A few of the almost 150 categories of NFRs [2] are maintainability, operability, performance, usability, and security. Understanding and implementing NFRs are critical to a software project. However, many developers overlook the importance of NFRs [3], spend less attention on NFRs due to financial and technical burden on the organization [4], or postpone implementation of NFRs until late in the development process


---

*In: F.B. Aydemir, C. Gralha, S. Abualhaija, T. Breaux, M. Daneva, N. Ernst, A. Ferrari, X. Franch, S. Ghanavati, E. Groen, R. Guizzardi, J. Guo, A. Herrmann, J. Horkoff, P. Mennig, E. Paja, A. Perini, N. Seyff, A. Susi, A. Vogelsang (eds.): Joint Proceedings of REFSQ-2021 Workshops, OpenRE, Posters and Tools Track, and Doctoral Symposium, Essen, Germany, 12-04-2021*

✉ rgnana1@students.towson.edu (R. K. Gnanasekaran); schakraborty@towson.edu (S. Chakraborty); jdehlinger@towson.edu (J. Dehlinger); ldeng@towson.edu (L. Deng)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

[5]. All of these significantly reduce the overall quality of software and lead to additional cost and efforts, and often result in frequent maintenance changes [6].

NFRs are often subjective and are dependent upon each organization's infrastructure capabilities. Consequently, the stakeholder's quality concerns that are encapsulated in NFRs become disorganized in the overall RE process, resulting in aspects related to NFRs being scattered across specification artifacts [5]. In addition, due to the fact that domain experts may not be available during the early RE phase, many NFRs are identified in the later phases of the software development process or not explicitly managed. Having engineers and experts go through and identify every NFR buried in large requirement documents is very tedious, time-consuming and error-prone. This often results in difficulties in isolating an NFR's functionality-related information and is also detrimental in developing an holistic understanding about them. Consequently, NFRs not only need to be identified, but also need to be consolidated for effective sense-making and implementation. Such consolidation often requires classifying NFR descriptions into pre-existing taxonomy. However, such classification is not a trivial task, as quality concerns are often expressed in natural language. Understanding them becomes a subjective process that depends upon one's interpretation of the language, which hinges on education, domain knowledge, expertise, etc. [5]. Due to these constraints, manual NFR classification is a labor intensive process and has potential human errors that can prove costly. Chung et al. [2] identified that natural language NFRs further conflict with each other in meaning and are often interconnected, making the manual classification process cumbersome. Automating this task can, therefore, increase the efficiency and effectiveness of the RE process. Machine-learning models for classification such as supervised neural networks offer an avenue for exploration.

This paper develops multi-class NFR classification using a type of Recurrent Neural Network (RNN), the Long Short-Term Memory (LSTM) model. Our best performing LSTM model achieves average precision, recall, and F1 score, across all target classes, for a testing data set at 84%, 85%, and 84%. The accuracy for LSTM on unseen testing data is 88%. Specifically, the contributions of this paper are: 1) The development of an approach using a LSTM to automatically classify NFRs; 2) An experimental study of the approach with widely used NFR data sets and report comparisons to other existing approaches; and, 3) An automated NFR analysis process to reduce manual efforts, human mistakes, and potentially increase software quality.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 introduces the approach designed in our study. Section 4 describes the experimental study used to evaluate our approach and analyzes the results. Section 5 identifies threats to validity and Section 6 concludes the paper and suggests future work.

## 2. Related Work

Categorization of NFRs have been carried out by several researchers over the past years. A comprehensive categorization of NFRs was performed by Chung et al. [2] where almost 150 categories of NFRs were identified. Roman [7] introduced 6 classes of NFRs as economic, interface, life-cycle, operating, performance, and political. Slankas and Williams [8] described 14 categories of NFRs. Based on these results, it is evident that the classification of NFRs depends upon the domain expertise of software engineers performing the task. The ambivalent nature of

this manual classification introduces errors thereby reducing the quality of software. This paper proposes an automated approach to classification of NFRs that can help mitigate these issues.

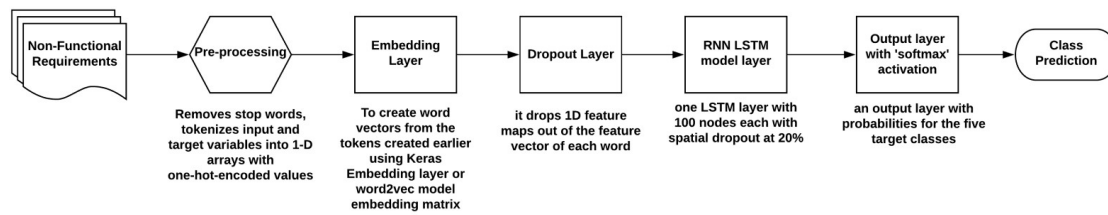
Several attempts have been made to classify NFRs automatically. In [5], Cleland-Huang et al. used information retrieval techniques to identify ‘indicator terms’ from NFRs for classification. Lu and Liang [9] attempted to classify the NFRs from the user application reviews using machine learning algorithms, including Naïve Bayes, Bagging, and a combined classification technique consisting of Bag of Words (BoW). Dekhtyar and Fong implemented a Convolutional Neural Network (CNN) model using Word2vec pre-trained algorithm for NFRs [10], however they were only able to perform binary classification and did not classify NFRs into different categories.

In prior work [11], we experimented with automated classification of NFRs using Artificial Neural Network (ANN) and CNN models. Also using the Support Vector Machine, [12] developed a supervised learning approach based on meta-data, lexical, and syntactical features. Their approach produced good results for binary classification with manually selected features but did not perform well with multi-class classification of NFRs. One of the drawbacks identified with traditional feed-forward deep learning neural networks, such as CNNs, is their inability to preserve representations from the previous input data for use in their learning process. This can lead to issues in classification problems with sequenced text, where there are significant relationships between a set of statements. While CNNs have been successful in handling computer vision data, RNNs have become the de-facto method for modeling text and audio data. This makes RNN as a potentially better candidate for NFR classification.

RNNs are a unique type of deep learning methods which are well suited for processing variable length sequences of input data. In this study, we aim to compare the results of our RNN LSTM with the CNN results from prior work and provide our findings. Prior to our study, there is very little research on NFR classification with RNN. In [13], Rahman et al. proposed a similar approach to the NFR classification problem by experimenting with the RNN architecture’s LSTM and Gated Recurrent Unit (GRU) variants. However, a potential issue with Rahman et al. is that their data set has 370 NFRs, which is quite small to complex neural networks, and may not result in reliably trained models. Our research rectifies this issue with the data set, by using a combination of two large public data sets containing almost 1,000 NFRs. In addition, the approach of Rahman et al. [13] used manually selected test data in the experiment without introducing how the test data were selected.

### **3. Approach**

This section explains the approach followed in classifying NFRs using RNNs. RNNs were initially developed to create recurrent links between networks which dynamically stored memory about prior representations of data by passing back internal states from hidden layers of previous steps to current steps, thereby allowing the networks to contextualize the training data for better results. The feedback loops in RNNs allow for information to persist. A traditional RNN model works better if the number of words in a sequence are nominal and loses more information if the number of words in the sequential input data increases. This is because of the weights assigned to a word that appears earlier in the time-series decreases since new words in the sequence gain more weights as the model moves forward. This impacts the unique contextual



**Figure 1:** RNN LSTM Model Development for NFR Multi-class Classification

learning behavior of the RNN model particularly if the words from further back on the current time-series are required to connect the information for classification.

Specifically, our research employs a particular form of RNN - the LSTM model. The LSTM structure is slightly complex within the recurring neural networks where there are four layers controlling the vector flow. In [14], Olah explains in simple words the architecture of the LSTM specific recurrent neural networks. The following steps are followed to construct the LSTM RNN model in our study: 1). Data set Pre-processing; 2). Word Vectorization Process; and, 3). RNN Model Construction. These steps are described next.

The first step requires pre-processing input text features of the NFR data set. The NFR data set has two features - an NFR requirement and the pre-labeled target class. The pre-processing involved five sub-steps: a). Conversion of the text to all lowercase to provide consistent input to the model and to remove additional noise; b). Replacing of certain mathematical symbols with spaces; c). Removal of special characters, using regular expressions; d). Removal of common *stopwords* using the nltk natural language processing library; and, e). Removal of *stopwords* unique to the data set. The second step is the tokenization and word embedding. As the deep learning model understands only numeric inputs, the text has to be broken down into “tokens,” through “tokenization.” We tokenize the NFR data using the Keras Tokenizer function<sup>1</sup>.

Another important aspect is the word embedding process. With the RNN model, where context of the words is important, one-hot encoding itself is not sufficient. To preserve relationships between words and to retain context, we perform a process called word embedding. The word embedding are added as a set of weight parameters that indicates the context, similarity and closeness between the words. We experimented with Word2vec<sup>2</sup> pretrained model embedding. Word2vec is an unsupervised model that uses Google’s pretrained word embedding algorithm where the model is trained on over 100 billion words. It seeks to embed words such that words often found in similar context are located near one another in the embedding space.

The final step is the RNN LSTM model construction. Figure 1 provides a sequential flow diagram of the construction of the RNN LSTM. As RNNs are recurrent in nature, to create a model which uses this architecture, we use TensorFlow Keras API’s Sequential model function to create a linear stack of sequential layers, i.e., input layer, hidden layer and output layer. The first input layer prepares word embedding. Next, a spatial dropout layer is added to sustain the meaning of all the words. Then the LSTM layer is added with nodes set to 100. After the

<sup>1</sup>Keras Tokenizer - [https://www.tensorflow.org/api\\_docs/python/tf/keras/preprocessing/text/Tokenizer](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer)

<sup>2</sup>Word2vec official link - <https://code.google.com/archive/p/word2vec/>

LSTM layer is created, the final output layer is constructed an activation function chosen to be “softmax.”

## 4. Experimental Evaluation and Results

### 4.1. Data Set

We use two large, publicly available data sets with pre-labeled NFRs: The International Requirements Engineering Conference’s 2017 Data Challenge data set<sup>3</sup> and the Predictor Models in Software Engineering Data set (PROMISE)<sup>4</sup>. Both data sets contain NFRs and each requirement’s class so we combined the two data sets, resulting in 1,165 unique NFRs falling into 14 categories. By combining the two data sets, we aim to address poor data availability issues raised in prior studies and also to eliminate possible biases that may exist in the data sets. Since 9 of the 14 categories have a small number of entries, only 5 categories are used in this study, which results in a total of 914 NFRs consisting of the following categories: maintainability (137), operability (153), performance (113), security (354), and usability (157).

### 4.2. Experiment Setup

For setting up the experiment, we compile the model using the Keras Compile<sup>5</sup> function by passing parameters with a specific loss function known as “categorical\_crossentropy,” which is generally used in multi-class classification problems and an optimizer function called “adam,” which is derived from “adaptive moment estimation.” To study the performance of the model, we include a set of commonly used metrics – *precision (P)*, *recall (R)*, *F1-score (F1)* and *area under the curve (AUC)* into the compile function to produce custom results for these metrics in addition to the accuracy metrics. After the model is constructed, the next step in the process is to train and test the model.

We split the data set into a 90% training and validation set and a 10% testing data set. The model is trained and cross-validated using the 90% data set. To make the model perform effectively, we train the model through cross-validation with “k” number of folds. This allows the model training to weigh each instance equally so that over represented classes don’t get too much weight. Specifically, we performed a stratified *k*-fold cross-validation on the training data set.

### 4.3. Results Analysis

The experiment includes tuning different hyper-parameters and other features. Table 1 lists the results using Word2vec embedding and shows values for the metrics obtained as a result of the *k*-fold cross-validation process and the testing on unseen test data. The *k*-fold cross validation results show the average results and standard deviation values of these metrics from the 10-fold training. The testing results show the results obtained by running the cross validated model on

---

<sup>3</sup>[http://ctp.di.fct.unl.pt/RE2017/pages/submission/data\\_papers/](http://ctp.di.fct.unl.pt/RE2017/pages/submission/data_papers/)

<sup>4</sup><http://promise.site.uottawa.ca/SERepository>

<sup>5</sup>Keras Metrics and Compile function - <https://keras.io/api/metrics/>

**Table 1**

Results from RNN LSTM with word2vec Provided Embedding Weight Matrix

Batch Size	<i>k</i> -fold Cross Validation Averages (%)				<i>k</i> -fold Cross Validation Standard Deviation (%)				Testing values (%)				
	P	R	F1	AUC	P	R	F1	AUC	P	R	F1	AUC	Accuracy
<b>10</b>	86.12	79.68	82.75	95.61	3.17	3.29	2.88	1.91	91.03	77.17	83.53	96.7	82.6
<b>30</b>	<b>84.18</b>	<b>77.26</b>	<b>80.56</b>	<b>94.8</b>	<b>3.67</b>	<b>4.61</b>	<b>4.13</b>	<b>1.73</b>	<b>91.36</b>	<b>80.43</b>	<b>85.55</b>	<b>97.5</b>	<b>88</b>
<b>60</b>	83.17	71.92	77.04	94.95	4.58	7.14	5.67	1.55	90.54	72.83	80.73	97.1	85.87
<b>Legend:</b> Precision (%) = P				Recall (%) = R			F1 Score (%) = F1			Area Under Curve (%) = AUC			

the test data set. The last column in Table 1 lists the classification accuracy calculated by the model evaluation process. For hyper-parameters, we configure the batch size to be 10, 30 and 60, embedding layer dimensions set at 300, epoch to be 50, LSTM layer nodes at 100. We highlight the results with highest classification accuracy associated each hyper-parameter tuning. The table also shows the *k*-fold cross-validation results for the standard deviation values for the precision, recall and F1-score averages from each run. The results show that the RNN word2vec model with batch size=30 achieves the highest accuracy at 88%.

Using the best performing model with the best hyper-parameters configuration, five individual tests were performed. Table 2 and 3 shows the results from five independent test runs of this highest performing Word2vec model with same data and tuning parameters to avoid any accidental stochastic errors. The tables list the values of metrics being evaluated: precision, recall, F1-score, and AUC for each of the 5 target NFR classes from these 5 independent test runs. These values were calculated for each class by analyzing the confusion matrix for each and identifying the true positives, true negatives, false positives, and false negatives.

The results show the performance metrics of each of the target NFR classes. Overall, the third and fourth test runs resulted in better classification accuracy, which is calculated by the model based on the true predictions on the test data. However, in imbalanced data sets, the classification accuracy calculated by the Keras in-built model evaluation<sup>6</sup> process may not provide clear results. Thus, looking at the individual precision, recall, and F1-score values by each class provides a better understanding of the results. Table 3 shows the average values of each metric calculated across all classes and the classification accuracy reported by the model's evaluation process. The table shows the average precision values range between 81%-87%, recall between 80%-87% with an F1-score between 80%-87% across these tests. The best classification accuracy was 88% found in Test Run # 3 and 4. Test Run #1 had the best average precision and recall values. In Test run # 2, Operability class shows a 100% recall with only 8 test entries. In Test runs # 2, 4 and 5, Performance class shows a 100% precision and recall with only 12 entries, respectively, which indicates that the model performs well with imbalanced classes.

Table 4 shows a comparison of the average metrics results and classification accuracy between our testing data and prior RNN work's testing data results. These results demonstrate that our RNN LSTM model performed better on all performance metrics with average increased classification accuracy of at least 15% than the reported results. This indicates that our model produced consistent results between independent test runs. A similar inference can be made for

<sup>6</sup>Keras model training - [https://keras.io/api/models/model\\_training\\_apis/](https://keras.io/api/models/model_training_apis/)





## 5. Threats to Validity

The RNN LSTM model trained and tested in this work faces threats rising from current probabilistic mechanisms in dealing with multi-category classification problems. As neural network models usually output real numbers as values for their classes, a transformation layer, such as the “softmax” function at the end of the model, converts them into values as probabilities between 0 and 1. NumPy library’s *argmax* identifies the class with the highest probability and assigns it as the observation’s target class. However such an assignment based on highest probability has an issue. Post-hoc manual analysis of the results indicate the reason behind this issue. Manual verification showed us that 11 out of the 92 tested NFRs got mis-classified by the model. For 7 of the 11 NFRs, the second highest probability predicted by the model belonged to a class originally labeled in the data set. Of these, in 4 of the 7 NFRs the predicted classes had highest probability scores of less than or equal to 50%, and the second highest probabilities were numerically very close. This suggests that classification based on highest probability without use of weights or confidence intervals pose an internal threat to validity in such multi-category classification problems.

One important external threat to the validity of the results from this study is with the labeling of the data set. As the NFRs in our data set are labeled manually, it is possible that the labeling process may not be accurate. Instances such as mislabeling, missing information, unavailability of required information could lead to the model misinterpreting the words from the beginning. One of the NFR statements in the data set was - “*Data is validated for type, length, format, and range*”. This was labeled as “Security” class. However it is not very clear why this NFR was labeled as “Security”. Such subjective ambiguity in labeling NFR has been noted in previous research [15]. In [15], it was reported that five security experts individually could not identify more than 50% of the security requirements with the information provided in the data set. Such issues reduce confidence in the accuracy of a data set and can potentially impact the performance of a neural net model.

## 6. Conclusion and Future Work

This paper introduces an LSTM-based approach to automatically classify NFRs and the results from our RNN LSTM model are better than prior existing works (e.g., [11, 12, 13]). Even with these better results, we still face threats due to the behavioral nature of the RNN LSTM’s learning process. Part of this is also due to the fact that the model takes each word by word as input to learn the context. To overcome this issue, we will perform research experiments on advanced natural language processing methods using popular pre-trained language representation models, such as Bidirectional Encoder Representations from Transformers (BERT)<sup>7</sup>. One advantage of using this model is the ability to process embedding at a sentence level rather than a word level, thereby treating words in the sentences as “WordPieces.” In addition, as this study focused only on categories of NFRs, in future work we will include FRs and assess the model’s performance. Source code for this project can be found on GitHub<sup>8</sup>.

---

<sup>7</sup>[https://huggingface.co/transformers/model\\_doc/bert.html](https://huggingface.co/transformers/model_doc/bert.html)

<sup>8</sup>[https://github.com/rgnana1/NFR\\_Classification\\_RNN\\_LSTM](https://github.com/rgnana1/NFR_Classification_RNN_LSTM)



## References

- [1] N. K. Sethia, A. S. Pillai, The effects of requirements elicitation issues on software project performance: An empirical analysis, in: C. Salinesi, I. van de Weerd (Eds.), *Requirements Engineering: Foundation for Software Quality*, Springer, 2014, pp. 285–300.
- [2] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, *Non-functional requirements in software engineering*, volume 5, Springer Science & Business Media, 2012.
- [3] R. R. Maiti, F. J. Mitropoulos, Capturing, eliciting, predicting and prioritizing (CEPP) non-functional requirements metadata during the early stages of agile software development, in: *SoutheastCon 2015*, 2015, pp. 1–8. doi:10.1109/SECON.2015.7133007.
- [4] M. Younas, D. N. A. Jawawi, I. Ghani, M. A. Shah, Extraction of non-functional requirement using semantic similarity distance, *Neural Computing and Applications* 32 (2020) 7383–7397. doi:10.1007/s00521-019-04226-5.
- [5] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, Automated classification of non-functional requirements, in: *Requirements Engineering*, volume 12, Springer-Verlag New York, Inc., 2007, pp. 103–120. doi:10.1007/s00766-007-0045-1.
- [6] R. Berntsson Svensson, T. Gorschek, B. Regnell, *Quality Requirements in Practice: An Interview Study in Requirements Engineering for Embedded Systems*, in: M. Glinz, P. Heymans (Eds.), *Requirements Engineering: Foundation for Software Quality*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 218–232.
- [7] Roman, A taxonomy of current issues in requirements engineering, *Computer* 18 (1985) 14–23. doi:10.1109/MC.1985.1662861.
- [8] J. Slinkas, L. Williams, Automated extraction of non-functional requirements in available documentation, in: *Proc. 1st International Workshop on Natural Language Analysis in Software Engineering*, IEEE, 2013, pp. 9–16.
- [9] M. Lu, P. Liang, Automatic Classification of Non-Functional Requirements from Augmented App User Reviews, *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering - EASE'17 (2017)* 344–353.
- [10] A. Dekhtyar, V. Fong, RE Data Challenge: Requirements Identification with Word2Vec and TensorFlow, 2017. doi:10.1109/RE.2017.26.
- [11] C. Baker, L. Deng, S. Chakraborty, J. Dehlinger, Automatic multi-class non-functional software requirements classification using neural networks, in: *Proceedings - International Computer Software and Applications Conference*, volume 2, 2019.
- [12] Z. Kurtanovic, W. Maalej, Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning, in: *Proc. IEEE 25th International Requirements Engineering Conference*, IEEE, 2017, pp. 490–495. doi:10.1109/RE.2017.82.
- [13] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, M. S. Siddik, Classifying Non-Functional Requirements Using RNN Variants for Quality Software Development, in: *Proc. eedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 25–30.
- [14] Understanding LSTM Networks – colah’s blog, 2015. URL: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [15] 25th IEEE International Requirements Engineering Conference, 2017. URL: [http://ctp.di.fct.unl.pt/RE2017/pages/submission/data\\_papers/](http://ctp.di.fct.unl.pt/RE2017/pages/submission/data_papers/).