A BERT-based transfer learning approach to text classification on software requirements specifications

Derya Kici ^{†,*}, Garima Malik [†], Mucahit Cevik[†], Devang Parikh[‡], Ayşe Başar[†]

[†] Data Science Lab, Ryerson University, Toronto, Canada

[‡] IBM, Toronto, Canada

Abstract

In a software development life cycle, software requirements specifications (SRS) written in an incomprehensible language might hinder the success of the project in later stages. In such cases, the subjective and ambiguous nature of the natural languages can be considered as a cause for the failure of the final product. Redundancy and/or controversial information in the SRS documents might also result in additional costs and time loss, reducing the overall efficiency of the project. With the recent advances in machine learning, there is an increased effort to develop automated solutions for a seamless SRS design. However, most vanilla machine learning approaches ignore the semantics of the software artifacts or integrating domain-specific knowledge into the underlying natural language processing tasks, and therefore tend to generate inaccurate results. With such concerns in mind, we consider a transfer learning approach in our study, which is based on an existing pre-trained language model called DistilBERT. We specifically examine the DistilBERT's ability in multi-class text classification on SRS data using various fine-tuning methods, and compare its performance with other deep learning methods such as LSTM and BiLSTM. We test the performance of these models using two datasets: DOORS Next Generation dataset and PROMISE-NFR dataset. Our numerical results demonstrate that DistilBERT perform well for various text classification tasks over the SRS datasets and shows significant promise to be used for automating the software development processes.

Keywords: Software requirement specifications, NLP, transfer learning, text classification, DistilBERT

1. Introduction

In the Software Development Life Cycle (SDLC), Software Requirements Specifications (SRS) play an important role by serving as a tool to communicate user requirements to software developers and other stakeholders. SRS are text documents where the key features, constraints, and functions of a software product are described. These documents must be written according to predefined standards so that all stakeholders such as users, analysts, and developers agree on the meaning of the specifications. SRS are also indicators for testing the quality and acceptability of the product or process at the end of the project. The criteria for the success of a software project are determined by how well the final product reflects the SRS documentation. Statements that are open to misinterpretation, ambiguous explanations, or imprecise inferences may result in critical failure later in the project. Accordingly, a clear and consistent understanding of these specifications requires consistent use of contextual terminology in that particular domain. Furthermore, the success of subsequent phases in SDLC depends on well-defined requirements and their accurate implementation; and failure to do so might cause delays and additional costs.

Although there are efforts to create a common language in the composition of the SRS such as controlled natural languages, and automated systems [1], the natural language is still preferred as the primary tool for this task. As such, it is important to design and

* derya.kici@ryerson.ca

implement new methods (e.g. using NLP, and deep learning techniques) that can automatically process SRS documents, and extract useful information such identifying entities and detecting anomalies.

Substantial amount of research have been dedicated to requirement analysis such as classifying and prioritizing requirements tasks using NLP [2–6]. However, most automated solutions made use of information retrieval and supervised machine learning techniques for the NLP tasks [6]. These techniques often ignore the semantics of software artifacts or ignore integrating domain knowledge into this process and accordingly tend to give inaccurate results. On the other hand, more advanced deep learning methods are capable of providing the reasoning behind a sentence that uses multiple levels of representation. Recent developments in NLP and deep learning, such as Convolutional Neural Networks (CNN) and Bidirectional Transformers, render a deeper understanding of the texts possible with automated systems. Deep learning methods has the ability to learn the rationale behind the data using a multi-level structure. The lower levels consist of general information about the data, and the deeper levels describe the more complex relationships between the words. These deep learning techniques can successfully address a number of problems raised by different types of ambiguities in a text document.

Shah and Jinwala [7] list various forms of *ambiguity* in SRS documents as follows:

- *Lexical ambiguity* of a word occurs when a word has several different meanings.
- *Syntactic ambiguity or structural ambiguity* in a sentence arises when the sentence can be parsed in more than one way, resulting in multiple grammatical structures, each of which providing different meanings.
- *Semantic ambiguity* arises in a sentence when the predicate logic of a sentence can lead to multiple interpretations although there is no lexical, syntactic, or structural ambiguity.
- *Language errors ambiguity* stems from a grammatically incorrect structure that is understood and possibly in different ways due to error.
- *Pragmatic ambiguity* deals with the relationship between the interpretations of a sentence and its context.

Another major challenge in using deep learning models for various NLP tasks in software engineering domain is the shortage of textual data. However, even with sufficient data, training from scratch still requires a significant training time and computational resources. *Transfer learning* proposes a solution to such reservations by providing pre-trained models that enable fine-tuning with the customized data. The shortcoming in classical approaches, built on an isolated learning paradigm, is that it performs ideally for well-defined and narrow tasks. To alleviate this, transfer learning extends the classic approach by using multitask learning with pre-trained models with domain adaptation and better generalization features [8].

In our analysis, we focus on IBM Rational DOORS Next Generation product (in short DOORS), which provides a scalable solution for optimizing communication, collaboration, and requirements validation [9, 10]. DOORS enables users to capture, monitor, analyze and manage changes in software requirements while maintaining compliance with regulations and standards. Integrating machine learning models that are able to process the textual information for various purposes such as classifying the type, severity and priority of a requirement has potential to significantly contribute to the DOORS product quality and help DOORS become the best available tool for software developers. For instance, accurate prediction of requirement type can save significant resources as it can be used to streamline the software development processes. Besides, the users who create the SRS documents might not always be knowledgeable about the type of the requirement. In addition, automatically identifying the priority and severity reduces the reliance on manual operations.

In this paper, we perform a comparative analysis with various text classification models over the SRS documents. Main contributions of our study can be summarized as follows.

- We study an important text classification problem in software requirements domain where the objective is to accurately predict the type, priority and severity associated with a given SRS document. To the best of our knowledge, previous studies did not consider text classification methods for this task.
- We use DistilBERT, a pre-trained general-purpose language representation model, on SRS documents. While DistilBERT was not specifically trained with software engineering domain-specific data, its high performance on generic text classification tasks makes it a suitable model for our purposes. To the best of our knowledge, previous works in software engineering domain that involve text classification tasks have not considered DistilBERT. Accordingly, empirical evaluation of the DistilBERT over software engineering datasets might lead to greater adoption of the model.
- We provide a detailed numerical study and compare the performance of DistilBERT against popular deep learning methods, namely, LSTM, and BiLSTM. We also consider GloVe and ELMo embeddings for LSTM and BiLSTM models, and empirically show the performance gains associated with pretrained models and word embeddings. We note that many NLP tasks in software engineering domain involve small-sized datasets, and our analysis contributes to a better understanding of the benefits of transfer learning.

The rest of the paper is organized as follows: Section 2 provides an overview of NLP methodologies, including LSTM, BiLSTM, and DistilBERT. We also present relevant studies in software engineering with a specific focus on text classification in SRS. In Section 3, we summarize the employed methodologies, dataset characteristics and our experimental design. Section 4 presents the results of our numerical study including performance comparison of multi-class text classification models for various text classification tasks. Finally, Section 5 provides concluding remarks and future research directions.

2. Literature Review

NLP research involves using the knowledge and methods from multiple disciplines (e.g. computer science, artificial intelligence, and computational linguistics) to create automated analysis and techniques to represent human language [11]. The primary goal is to develop algorithms that give machines the ability to process, understand and produce natural language similar to humans. Over the years, a variety of applications has been developed in this area, e.g., speech recognition, language translation, text classification and summarization. However, the biggest challenge for machines to imitate human interaction with natural language is the dependence of natural language on context. People can take real-world situations and conditions into account when interpreting and producing language, however, automated systems have not yet perfectly developed this ability to handle the corresponding context complexity [12].

To address such issues, NLP algorithms have been evolved from the simple statistical language models that only calculate the probability distribution over sentences without any focus on semantic structures to the parts-of-speech tagging and name entity recognition that aim to reveal the meaning of the sentences considering the semantic components of the sentences [13]. Deep learning have been a popular approach for various NLP tasks including text classification, achieving state-of-the-art results for various problems [14]. In particular, Recurrent Neural Network (RNN) models, which consider text as a sequence of words, have the ability to capture word dependencies and structure of the text. In particular, Long

Short-Term Memory (LSTM) networks, a variant of RNNs, are designed to better capture the long term dependencies by using memory cells in the network and have been widely adopted for text classification. Another RNN variant, Bidirectional LSTMs (BiLSTMs) contain two networks: first one to access past information in forward direction, and second one to access future information in reverse direction. This way BiLSTMs can understand the semantics and context of the text better.

Recent deep learning approaches mostly rely on transfer learning and pretrained models. Bidirectional Encoder Representations from Transformers (BERT) combines bidirectional transformers and transfer learning with the objective of creating state-of-the-art models for a wide range of NLP tasks [15]. BERT has several variations such as RoBERTa, ELECTRA, DistilBERT, and ALBERT [14]. In our analysis, we use DistilBERT, which includes 66 million parameters that make it 40% smaller and 60% faster than BERT base [16].

Text classification is a practical NLP task in computer science and linguistics research. Software engineering is also a prosperous field for text classification with various types of textual information in the domain. For instance, Bacchelli, Dal Sasso, D’Ambros, and Lanza [17] considered machine learning models to classify the content of emails related to the development of a software system that contain information about design choices and issues encountered during the development process. Khan, Syed, Khan, and Rafi [18] investigated the end users license agreement’s reliability by classifying the contents as “benign” or “malicious” by eight different supervised classifiers including logistic regression, SVM, and Naive Bayes models. Dias Canedo and Cordeiro Mendes [19] compared the classification performance of Logistic Regression (LR), SVM, Multinomial Naive Bayes (MNB), and k-Nearest Neighbors (kNN) over SRS datasets. Similarly, Ott [20] investigated SRS documents using Naive Bayes and SVM models to determine whether requirements with related information spread over many sections of many documents. Dollmann and Geierhos [21] implemented binary classification through decision tree, Naive Bayes, SVC, and NuSVC as well as ensemble methods such as Random Forest and Ada Boost in the development of requirements extraction and classification tool for identifying, and semantically annotating the on-topic parts of requirement descriptions.

Asif, Ali, Malik, Chaudary, Tayyaba, and Mahmood [22] considered the problem of classifying functional and non-functional requirements in SRS documents, and experimented with different classifiers such as SVM, kNN, and Naive Bayes. Another research that investigate the same classification problem was conducted by Tiun, Mokhtar, Bakar, and Saad [23]. Researchers compared the performance of different word embeddings (e.g., Doc2Vec and FastText) and three types of traditional machine learning classifiers (SVM, Naive Bayes, and LR) against the CNN classifier. Hussain, Ormandjieva, and Kosseim [6] employed a decision-tree-based text classifier to build an automatic detection system to identify ambiguities in SRS documents. In addition, some researchers explored SRS documents through information retrieval-based approaches [4, 24].

Several recent studies on classification problems involving SRS documents focused on deep learning models. Navarro-Almanza, Juarez-Ramirez, and Licea [5] employed CNNs to identify the functional and non-functional requirements. Onyeka, Anu, and Varde [3] developed a CNN-based approach to detect implicit requirements from complex SRS big data. Despite the abundance of such studies involving SRS documents, which showcase the better performance of deep learning architectures over the traditional classifiers, only a few studies consider transfer learning-based approaches in the software requirement domain. Recently, Hey, Keim, Koziolk, and Tichy [25] studied the problem of classifying functional and non-functional SRS documents using transfer learning-based approaches such as NoRBERT. They concluded that NoRBERT improves the prediction performance considerably.

3. Methodology

In this section, we first describe our SRS dataset and present an exploratory data analysis on two dataset. Then, we provide details on classification models and experimental settings for our numerical study.

3.1. Dataset

In this study, our SRS data was obtained with the Dynamic Object Oriented Requirements System (DOORS) Next Generation, a requirement management tool developed and marketed by IBM [10], which is widely used by engineers worldwide. Our SRS dataset consists of 83,837 instances and 213 features. However, we only examined the summary of the specifications to classify the categorical features as the type, severity and priority of the requirements.

In our dataset, among the 83,837 text documents, the longest document contains 77 words while the shortest one has only 1 word. These documents also consists of minimum 1 and maximum 387 characters. Figure 1a and 1b present the distribution of the words and characters in our SRS dataset. Distribution of words and characters are right-skewed. The mean and median of number of characters in SRS documents are 61 and 57, respectively, and the mean and median of number of words in SRS documents are 11.35 and 10, respectively.

In the first step of pre-processing, we applied tokenization on the text by dividing the sentences into individual words. We removed the stop words from the corpus as the presence of these words in text data might adversely affect the analysis. By doing so, we built our unique vocabulary of SRS documents. In order to preserve the semantic structure of the words in the original SRS documents created in software development processes, we avoided the lemmatization or stemming techniques. We also investigated for bigrams and trigrams, which are contiguous sequence of two and three words that are often used together in the text documents. Figure 1c, and 1d show the top 10 of the bigrams and trigrams, respectively.

We next focused on the classes considered for document classification as the second step of data preprocessing. We considered three characteristics for classification of SRS documents, namely, *Type*, *Priority* and *Severity*. Each of these categories contains a different number of classes that are not evenly distributed as shown in Figure 2.

There are a total of 21 different classes in the *Type* feature. Among these 21 classes, *Defect* is the first and *Task* is the second most dominant classes. The ratio of *Defect* to *Task* is 2.37 while its ratio to the sum of all other instances other than *Task* is 2.24. We also combined twelve classes with the lowest frequency (260 and less) in a new class with label *Other*. The sum of the frequencies of all twelve classes is 1,202, and the closest frequency belongs to the *Plan Item* class with 1,224. As a result, we obtained 8 classes to apply text classification for the *Type* category.

There are four classes of concern for the *Priority*; unassigned, high, medium and low. Priority of the most documents remained unassigned, which is excluded from our analysis. Hence, for this category, we have three class labels of interests (high, medium, low). For *Severity*, there are 6 classes, which are normal, major, minor, blocker, critical, and undecided. After excluding a few instances for “undecided” label, 5 class labels remain for the classification task.

3.2. Deep learning models

We provide the list of deep learning models and the word embeddings below.

- **LSTM:** LSTM networks [26] have been found to be able to learn long term dependencies and patterns across natural language text. The major advantage of LSTMs

Figure 1. Exploratory data analysis results on DOORS dataset

over other RNNs is that their usage of forget and update gates allow them to regulate and maintain constant information flow, effectively providing protection from the issues of vanishing and exploding gradients.

- **BiLSTM:** The bidirectional variant of LSTM models process the input data once from beginning to the end, and once from end to beginning [27]. BiLSTM explores the deeper semantics of the word structure. Forward and backward training provides an additional understanding of contextual dependency present in the tokens.
- **DistilBERT:** DistilBERT transformer is a general purpose language representation model that is smaller, faster, and lighter than BERT [16]. In this model, through knowledge distillation in the pre-training phase, the size of the BERT model is reduced by 40%.
- **Embeddings:** Kernel embeddings help generating numerical form of vectors without consideration of the context of input. Similarly, GloVe embeddings create a distributed representation of the input text using pre-trained weights. That is, a word is converted to same feature vector irrespective of its syntactical form and context. To acknowledge the syntactical and semantics of words or tokens, Peters,

Figure 2. Class distribution of categorical features in the DOORS SRS dataset

Neumann, Iyyer, Gardner, Clark, Lee, and Zettlemoyer [28] proposed deep contextualized word embeddings called ELMo, which stands for Embeddings from Language Model. These embeddings were inspired from the language model architecture where each word in the input text will be converted to a appropriate feature vector and same word can have different embeddings depending upon the context. In a context aware embedding model, aim is to augment the token with its context and output of the neural network is the function of token and its context ($y = f(\text{word}, \text{context})$). The architecture of ELMo embeddings is similar to vanilla language models. Each token is passed through forward and backward layers of LSTM or GRU.

3.3. Experimental setup

In our analysis with DistilBERT, we instantiated the transformer, namely distilbert-bert-uncased model, and modified the architecture for the multi-class text classification of SRS documents by using the *DistilBertForSequenceClassification* class in the transformer library. Accordingly, we fine-tuned the weights using our labeled SRS dataset. In our experiments, we trained our classifier with batch sizes of 32 for 15 epochs. The dropout probability is set to 0.1 for all the layers and AdamW optimizer is used with 1e-5 learning rate.

In computationally expensive models such as BERT, the training process is very resource intensive, especially for fine tuning of all the model layers. However, we can alleviate this by controlling the line length of the input text. Line length is the sum of the number of symbols in our input text, and two special symbols are used to mark the beginning ([CLS]) and the end ([SEP]) of a text string. As an input, we tokenized each SRS documents with the DistilBertTokenizer. After tokenization of the text, we encoded it into the corresponding numerical values for each symbol. Following this, we set the maximum sequence length to 77, as the summaries of requirements are short texts. Token sequences are either padded with zero values or truncated to the maximum length.

Finally, we added [CLS] and [SEP] as special token IDs to mark the beginning and end of each sequence. To fine-tune our model, we need two arrays as input; (1) an array of token IDs (2) an array of a corresponding binary mask sequences referred to as the attention mask in the BERT model specification. The length of each attention mask is the same as the length of the corresponding input row. When the corresponding icon is a pad icon, this length is 0, otherwise 1.

For training purposes, we consider 80% of data as training data to update the weights in the fine-tuning phase, 10% for validation to measure the out-of-sample performance of the model during training, and 10% for the final testing to measure the out-of-sample performance after training. To prevent over-fitting, we use stratified sampling to select 0.8, 0.1, and 0.1 portions of SRS document from each class for train, validation, and test.

In LSTM experiments, we applied early stopping that monitors if the loss is increasing for 5 consecutive epochs then training is stopped to decide the epoch values for all the models. For the LSTM model, we used 256 units and epoch number ranging from 10 to 100 with early stopping. With respect to the kernel output dimensions, lower values negatively impacted the accuracy and F1-score, therefore we decided to keep it as 100. Regarding BiLSTM, the ideal number of units is identified as 128 since higher values impacted the score negatively. As dropout parameter, we kept standard value as 0.2, since the lower values for kernel output embedding dimension did not provide significant performance gains. For BiLSTM + GloVe, the GloVe dimensions were determined as 300 that provided the best F1-score and for the epoch number, we used 20 that proved to be ideal number as afterwards the loss increased. In the experiments with BiLSTM + ELMo, we relied on the default ELMo output vector of dimension 1,024 with LSTM units as 128. We performed k-fold cross validation with stratified k fold to have a uniform distribution in test set in the experiments for all the models mentioned above. Table 1 presents hyperparameters for all the models.

Table 1. Hyperparameter values used in training the models

Model	Settings
DistilBERT	hidden_size=768, max. seq. length = 77, batch size = 32, learning rate = 1e-5, epochs = 5
LSTM	hidden_units = 256, epochs = 20, Kernel embedding = 100
BiLSTM	hidden_units = 128, epochs = 10, recurrent-dropout = 0.2, Kernel embedding = 100
BiLSTM+Glove	Glove dimensions = 300, hidden_units = 128, epochs = 20, dropout = 0.2
BiLSTM+ELMo	ELMo output vector = 1024, hidden_units = 128, dropout = 0.2

4. Results

Our experiments are designed to investigate the effectiveness of deep learning models and transfer learning in SRS document classification. We compare DistilBERT, a transfer learning approach, LSTM and BiLSTM, which are among the two most prevalent deep learning methods used for text classification.

As the performance indicators, we rely on the standard classification metrics such as precision, recall, and weighted-average F1-score. Precision measures the success of the model

when the output prediction is positive by evaluating the ratio of correct predictions to all positive predictions. Recall evaluates the model based on its success on correctly predicting an observation in a positive class. The F1-score provides a better understanding of the model by taking harmonic mean of precision and recall. As such, we consider weighted-average F1-score as the most reliable metric in a dataset that consists of uneven class distribution.

Table 2 compares these accuracy, precision, recall, and weighted-average F1-score from the five models that we implemented to classify the Type category on DOORS data. We observe that DistilBERT performs better than all other models while the basic LSTM model provides the worst performance values. BiLSTM models perform better than their LSTM counterparts. In addition, BiLSTM + ELMo and BiLSTM + GloVe perform better than standalone models with kernel embedding layer. Also, the comparison of the embeddings indicate that ELMo perform slightly better than GloVe, which can be attributed to the fact that ELMo is a context aware embedding. We also provide the F1-scores obtained by using cross-validation as box plots in Figure 3, which confirms that DistilBERT is the best model for classifying Type category in our SRS dataset.

Table 2. Classification results for SRS type

Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
DistilBERT	80.79	80.00	80.79	80.00
LSTM	76.97	76.68	77.00	76.75
BiLSTM	77.58	77.37	77.59	77.31
BiLSTM + GloVe	79.71	79.33	79.71	79.18
BiLSTM + ELMo	79.80	79.48	79.75	79.33

Figure 3. F1-scores obtained from different models using cross-validation

We also examined the loss curves of the deep learning models for each experiment. In these loss curves, each fold loss obtained over each epoch is averaged. We did not observe particular pattern in model convergence behaviours. However, we found that all the models tend to converge within a few epochs.

As DistilBERT is the best performing model, to better understand its performance in terms of classifying type of SRS documents, we provided the confusion matrix in Figure 4. We observe that the model can separate all classes at different level of accuracy. Even though DistilBERT is successful in identifying JUnit, Story, Task Task, Maintenance, and Enhancement with accuracy higher than 80%, it fails to distinguish RFE from Enhancement. The model also performs poorly for identifying the types we grouped as “Other”.

Figure 4. Confusion matrix for type classification with DistilBERT (Support values: Enhancement 1105, Story 1043, Maintenance 423, J unit 356, Test Task 380, Plan Item 250, RFE 232, Others 222)

We also trained DistilBERT to classify Priority and Severity information associated with the DOORS SRS dataset. Table 3a shows that the models are much less effective in classifying Priority category compared to predicting Type category. DistilBERT performs particularly poorly for Priority prediction, which can be attributed to a relatively low number of training instances with valid class labels for Priority (i.e., due to excluding instances with “unassigned” labels as well as “nan” labels, which constitute 64% of the labels for Priority. On the other hand, the results of the experiments to classify Severity class as presented in Table 3b provides much better performance in comparison to Priority category. Similarly, we excluded instances with “undecided” and “nan” labels for the Severity category as well, which comprise only 0.15% of the total instances.

Lastly, we experimented with publicly available PROMISE-NFR dataset which contain 626 instances describing functional and non-functional requirements. The dataset contains 11 labels: availability, legal and licensing, look and feel, maintainability, operability, performance, scalability, security, usability, fault tolerance, and portability. Our experiments with the five models showed that DistilBERT and BiLSTM + ELMo performed similarly, with 74% F1-score, however, other models had less than 53% F1-score. The relative performances of these models on PROMISE-NFR dataset largely overlaps with our findings with DOORS SRS dataset.

5. Conclusion and future work

In this study, we examine the effectiveness of the DistilBERT transformer as a state-of-the-art transfer learning model for multi-class text classification over SRS documents.

Table 3. Classification results for DOORS SRS dataset Priority and Severity

(a) Priority				
Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
DistilBERT	61.57	60.00	61.57	60.00
LSTM	56.25	56.27	56.25	56.06
Bi-LSTM	57.92	56.44	57.92	57.03
Bi-LSTM - ELMO	58.45	57.22	58.62	57.44
Bi-LSTM - GloVe	60.37	58.61	60.06	57.28
(b) Severity				
Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
DistilBERT	75.53	68.00	76.00	71.00
LSTM	72.50	68.42	72.50	70.00
Bi-LSTM	73.93	68.75	73.92	70.55
Bi-LSTM - ELMO	74.11	69.30	74.05	70.91
Bi-LSTM - GloVe	75.57	70.92	75.57	70.83

We evaluate the effectiveness and performance of this model by comparing it with high performing deep learning models, namely, LSTM, BiLSTM, BiLSTM + GloVe, and BiLSTM + ELMo. As the results of our experiments suggest, DistilBERT over-performed the RNN-based models including the variants of the models with different embeddings, which can be attributed to the fine-tuning capabilities of the pretrained models such as DistilBERT. We note that the fine-tuning can be particularly useful for fields such as software engineering where the amount of labeled data is limited.

We provided comparative results of the models for the classification of Type, Priority, and Severity categories for DOORS data, and Type category for PROMISE-NFR dataset. Although the classification results for Priority and Severity categories as well as PROMISE-NFR requirement types did not provide high accuracy, they are still promising considering that all have multiple classes to predict. In order to develop a better understanding of the models with lower accuracy, we aim to conduct a deeper qualitative analysis such as investigation of errors made by the models as the next step. We also aim to investigate different model architectures with different capabilities such as a multi-label classification model as the next step of our study.

Among BERT models, we only considered DistilBERT, however different BERT-based transformers and fine-tuning strategies can be considered for our SRS classification tasks, which we left for future research. Furthermore, such BERT models, including DistilBERT, are pretrained with generic text datasets. Therefore, software engineering domain specific pre-trained models might achieve better results for the SRS classification tasks as well.

References

- [1] M. Duggal, N. Saxena, and M. Gurve. “SRS Automator-An Attempt to Simplify Software Development Lifecycle”. In: *2020 6th International Conference on Signal Processing and Communication (ICSC)*. IEEE. 2020, pp. 278–283.
- [2] M Haris, T. Kurniawan, and F. Ramdani. “Automated Features Extraction from Software Requirements Specification Documents as The Basis of Software Product Line Engineering”. In: *Journal of Information Technology and Computer Science* 5.3 (2020), pp. 279–292.
- [3] E. Onyeka, V. Anu, and A. S. Varde. “Identifying implicit requirements in SRS big data”. In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 6169–6171.

- [4] I. Ali, M. Asif, M. Shahbaz, A. Khalid, M. Rehman, and A. Guergachi. "Text categorization approach for secure design pattern selection using software requirement specification". In: *IEEE Access* 6 (2018), pp. 73928–73939.
- [5] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea. "Towards supporting software engineering using deep learning: A case of software requirements classification". In: *CONISOFT 2017*. IEEE. 2017, pp. 116–120.
- [6] I. Hussain, O. Ormandjieva, and L. Kosseim. "Automatic quality assessment of SRS text by means of a decision-tree-based text classifier". In: *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE. 2007, pp. 209–218.
- [7] U Shah and D Jinwala. "Resolving ambiguities in natural language software requirements: a comprehensive survey". In: *ACM SIGSOFT Software Engineering Notes* 40.5 (2015), pp. 1–7.
- [8] S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf. "Transfer learning in natural language processing". In: *Proceedings of the 2019 Conference of the North American Chapter of the ACL: Tutorials*. 2019, pp. 15–18.
- [9] E. Hull, K. Jackson, and J. Dick. "DOORS: a tool to manage requirements". In: *Requirements engineering*. Springer, 2002, pp. 187–204.
- [10] J. Dick, E. Hull, and K. Jackson. "Requirements Engineering in the Problem Domain". In: *Requirements Engineering*. Springer, 2017, pp. 113–134.
- [11] T. Young, D. Hazarika, S. Poria, and E. Cambria. "Recent trends in deep learning based natural language processing". In: *IEEE Computational intelligence magazine* 13.3 (2018), pp. 55–75.
- [12] B. S. Haney. "Patents for NLP Software: An Empirical Review". In: *Available at SSRN 3594515* (2020).
- [13] N. Ranjan, K. Mundada, K. Phaltane, and S. Ahmad. "A Survey on Techniques in NLP". In: *International Journal of Computer Applications* 134.8 (2016), pp. 6–9.
- [14] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao. "Deep learning based text classification: A comprehensive review". In: *arXiv preprint arXiv:2004.03705* (2020).
- [15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).
- [16] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).
- [17] A. Bacchelli, T. Dal Sasso, M. D'Ambrosio, and M. Lanza. "Content classification of development emails". In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE. 2012, pp. 375–385.
- [18] B. Khan, T. Syed, Z. Khan, and M. Rafi. "Textual analysis of End User License Agreement for red-flagging potentially malicious software". In: *ICECCE 2020*. IEEE. 2020, pp. 1–5.
- [19] E. Dias Canedo and B. Cordeiro Mendes. "Software Requirements Classification Using Machine Learning Algorithms". In: *Entropy* 22.9 (2020), p. 1057.
- [20] D. Ott. "Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements". In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer. 2013, pp. 50–64.
- [21] M. Dollmann and M. Geierhos. "On-and off-topic classification and semantic annotation of user-generated software requirements". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 1807–1816.
- [22] M. Asif, I. Ali, M. S. A. Malik, M. H. Chaudary, S. Tayyaba, and M. T. Mahmood. "Annotation of Software Requirements Specification (SRS), Extractions of Nonfunctional Requirements, and Measurement of Their Tradeoff". In: *IEEE Access* 7 (2019), pp. 36164–36176.
- [23] S Tiun, U. Mokhtar, S. Bakar, and S Saad. "Classification of functional and non-functional requirement in software requirement using Word2vec and fast Text". In: *Journal of Physics: Conference Series*. Vol. 1529. 4. IOP Publishing. 2020, p. 042077.
- [24] J. Cleland-Huang, R. Settini, X. Zou, and P. Solc. "The detection and classification of non-functional requirements with application to early aspects". In: *RE 2006*. IEEE. 2006, pp. 39–48.

- [25] T. Hey, J. Keim, A. Koziol, and W. F. Tichy. “NoRBERT: Transfer learning for requirements classification”. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE. 2020, pp. 169–179.
- [26] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [27] M. Schuster and K. K. Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.
- [28] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. “Deep contextualized word representations”. In: *arXiv preprint arXiv:1802.05365* (2018).