RE-BERT: Automatic Extraction of Software Requirements from App Reviews using BERT Language Model

Omitted due to the double-blind review process

ABSTRACT

Traditionally, developers restricted themselves to collecting opinions from a small group of users by using techniques such as interviews, questionnaires, and meetings. With the popularization of social media and mobile applications, these professionals have to deal with crowd users' opinions, who want to voice the software's evolution. In this context, one of the main related tasks is the automatic identification of software requirements from app reviews. Recent studies show that existing methods fail at this task, since review texts usually contain informal language, contain grammatical and spelling errors, as well as the difficulty in filtering out irrelevant information that has no practical value for developers. In this paper, we present the RE-BERT (Requirements Engineering using Bidirectional Encoder Representations from Transformers). Our method innovates by using pre-trained neural language models to generate semantic textual representations with contextual word embeddings. Our RE-BERT performs fine-tuning of the BERT model with a focus on the local context of the software requirement tokens. A statistical analysis of the experimental results involving 8 different apps showed that our RE-BERT outperforms three existing state-of-the-art methods.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**; **Requirements analysis**; • **Computing methodologies → Neural networks**;

KEYWORDS

app reviews, opinion mining, requirement extraction, requirement engineering, BERT, neural language models

ACM Reference Format:

Omitted due to the double-blind review process. 2021. RE-BERT: Automatic Extraction of Software Requirements, from App Reviews using BERT Language Model. In *Proceedings of ACM SAC Conference (SAC'21).* ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/xx.xxx/xxx_x

SAC'21, March 22-March 26, 2021, Gwangju, South Korea

ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Extracting web and social media requirements has become a new trend in Software Requirements Engineering (ERS) [4, 11, 12]. Traditionally, developers restricted themselves to collecting opinions from a small group of users by using techniques such as interviews, questionnaires, and meetings. With the popularization of social media and mobile applications, these professionals have to deal with crowd users' opinions, who want to voice the evolution of the software [13]. To create successful products quickly and with greater acceptance by users, developers need to take a proactive stance about the analysis of the feedback they receive from users [11].

App reviews have become one of the primary sources of user opinions about software. Users can express themselves through evaluation notes and feedback in free texts [3, 12]. Free text opinions can, for example, report defects, express your satisfaction with an application-specific feature, or request a new feature [7]. However, popular applications receive hundreds of thousands or even millions of reviews, and manual analysis of this large volume of information becomes an impractical task [11–13]. Therefore, several works were developed in the last decade to create tools to support this process [14]. Among several review-based software requirements engineering tasks, the correct identification of software requirements from app reviews is one of the main steps. The quality of the requirements extraction task will directly impact other important tasks, such as the polarity classification (positive, negative and neutral) of users regarding a software requirement.

The automatic extraction of requirements from app reviews is not a trivial task [14]. Among several challenges, we can highlight the i) use of natural (unstructured) and often informal language, neglecting grammatical and punctuation, using eccentric syntactic entities, ironic and sarcastic phrases [13]; ii) the same review has multiple aspects of the app, where different opinions are related to different features [7]; iii) large volume of noise present in the reviews, irrelevant information, which has no practical value for developers [9, 12]; iv) variety of app domains, where each domain has a set of particular features and vocabulary [6].

ERS approaches, proposed to minimize these problems, explore opinion mining from app reviews to automatically extract software requirements [6, 7, 9, 15]. However, recent studies comparing the performance of these approaches indicate that there are still major open challenges to extracting software requirements effectively [4, 12]. Most of these approaches are based on linguistic rules applied in the reviews. Such rules usually depend on part-of-speech (PoS) tools that are not suitable for informal texts, with spelling and grammatical errors. In addition, many requirements reported in user reviews depend on context to be properly identified, which is a limitation of methods based on linguistic rules.

In this paper, we present the RE-BERT method (Requirements Engineering using Bidirectional Encoder Representations from Transformers), which has four pillars. First, it is based on context-dependent language models. Therefore, it covers existing gaps in using rule-based approaches such as efficiency, accuracy, generalization, and dependence on external sources [1]. RE-BERT also covers gaps in strategies based on non-contextual word embeddings, such as the difficulty of dealing with ambiguity and semantic [5]. Second, RE-BERT uses a cross-domain training strategy, which allows training a model on pre-existing labeled data from different apps, so that the model can be used to extract requirements from another different domain and unlabeled review dataset. Third, we use a training strategy based on a local and global context [16]. In particular, the local context is able to identify words semantically more related to software requirements. Fourth, RE-BERT allows pre-training and fine-tuning. The pre-trained version is ready for use in any app reviews domain. The fine-tuning allows upgrading the pre-trained model's extraction capacity, thereby training with reviews of the specific domain, when these texts are available.

We carried out an experimental comparison of RE-BERT using the datasets and classification results previously reported by [4]. The RE-BERT performance was compared with ReUS [6], SAFE [9] and GuMa [7]. Our method obtained F_1 (harmonic mean between precision and recall) performance statistically superior to the other three methods in all eight app reviews datasets. In particular, it is worth noting that RE-BERT was promising in the exact matching evaluation scenario, in which the method must correctly identify all tokens of the software requirement. In terms of evaluation measure F_1, improvements ranged from 80% to 560%.

The rest of the paper is structured as follows: In Section 2, we present an overview of the approaches we use as a basis for comparison for the proposed method. In Section 3, we present the architecture of the proposed RE-BERT method. In Section 4, we present details of the experiments carried out and discuss the results obtained. The conclusions and directions for future work are presented in Section 5.

2 RELATED WORKS

An automatic app review analysis pipeline can involve several steps. The extraction of review requirements is responsible for locating aspects in the review text related to a specific requirement [9]. The review's classification organizes the reviews in essential classes for the developers, such as Bug Reporting, New Requirement, and Use Experience with Existing Requirement [11]. The clustering of requirements extracted from the apps review organizes similar requirements and can also define a hierarchical relationship between the extracted requirements [15]. Finally, the sentiment analysis of the requirements extracted from the reviews defines the polarity (positive, negative, or neutral) of the user's feedback [7]. The quality of the requirements obtained in the extraction stage is key to the other stages' success, so the proposed approach focuses on improving this stage's performance. Therefore, even if the approaches include other stages, we will restrict ourselves to the extraction stage's scope when presenting the related works.

GuMa [7] uses a collocation finding algorithm provided by the NLTK toolkit to perform requirements extraction. Collocations are the expressions of two or more words that correspond to a conventional way of referring things. In this context, the authors used a likelihood-ratio measure to find collocations that consist of two words in the reviews. Next, a filter was made to find the collocations, account only those collocations that appeared in at least three reviews, and that are less than three words (nouns, verbs, or adjectives) apart. In the end, were merged the collocations that had the same meaning.

SAFE [9] extracts features based on linguistic patterns, including 18 part-of-speech patterns and 5 sentence patterns. Authors make a manual analysis of the textual description to identify these patterns. In the first stage, the approach performs text pre-processing, including tokenizing a sentence review, eliminating noisy sentences, and removing unnecessary words from the relevant sentences. In the second and last stage, linguistic patterns are applied to each sentence to extract software requirements.

ReUS [6] explores linguistic rules composed of grammatical class patterns and semantic dependency relationships. The analysis and extraction of characteristics are done based on these rules. These two tasks are run out together. The approach extracts features and an opinion word that conveys the features' specific sentiments for each sentence.

In [4], the authors presented an experimental comparison among the SAFE, ReUS and GuMa approaches and in [8] the SAFE approach was compared with the CLAP approach. The common factor among all these approaches is that they use a rules-based information extraction strategy, where a subset of app reviews have been previously analyzed for common linguistic patterns related to the text passages that describe a software requirement. Subsequently, algorithms were trained to automatically identify these pre-defined patterns that represent the requirements. However, the literature points out limitations for rule-based approaches [1]:

- Efficiency: a textual expression can be represented by a large number of corresponding rules or by rules with high complexity;
- Accuracy: a large volume of rules can generate many conflicts, or a small number of rules may not be representative enough to extract the desired information;
- Generalization and lack of context: rules extracted by analyzing data from a specific domain generally do not meet the context of other domains; and
- Dependency: requires the use of external dictionaries to extract context information, e.g., identifying whether a token belongs to a specific type, such as titles, locations, and organizations.

In [10], the problem of extracting requirements from social software development networks was mapped to the task of extracting named entities. An approach based on the BI-LSTM model (Bidirectional-Long Short Term Memory) with word embedding was proposed using Word2Vec. This approach was also used in the context of app reviews [15]. Word embedding-based approaches have advantages over Bag-of-word based approaches with word frequency counting (TF-IDF). However, word2vec-based approaches have semantic limitations generated by their static representation of the word embeddings [5]. Also, because word2vec is based on architecture without an attention mechanism, it has an efficiency

problem and difficulty in determining word dependency in long texts [5].

Given these limitations, our work proposes a new approach for requirements extraction based on Bidirectional Encoder Representations from Transformers (BERT). The BERT [5] model presented promising results, surpassing rule-based approaches, static word embedding approaches (word2vec), and in some natural language tasks, even human performance. BERT model overcame the state-of-the-art for natural language processing in many tasks.

3 THE RE-BERT METHOD

The extraction of software requirements from app reviews can be defined as a token classification problem. We used the BIO format (short for beginning, inside, outside) to structure the training set reviews, as shown in Figure 1. Note that the tokens for each sentence in the training set are labeled with **class B**, which indicates that the token represents the beginning of a software requirement; **class I**, which indicates that the token is inside a software requirement; and **class O**, which indicates that the token is outside a software requirement in the sentence.

The	app	crashes	when	I	try	to	share	photos
O	O	O	O	O	O	O	B	I

Figure 1: Example of app reviews labeling in OBI format for software requirements extraction.

Given a sequence of tokens representing an app review $\mathbf{x} = (x_1, x_2, ..., x_T)$, the goal is to find a sequence of labels $\mathbf{s} = (s_1, s_2, ..., s_T)$, one label for each token, according to Equation 1.

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}} p(\mathbf{s}|\mathbf{x}) \tag{1}$$

A promising strategy for estimating the probability function $p(\mathbf{s}|\mathbf{x})$ is via sequence labeling models [2]. Sequence labeling methods are often based on a language model that associates a probability of occurrence for a given sequence of tokens $\mathbf{x} = (x_1, x_2, ..., x_T)$. In practice, the objective is to compute the likelihood of a token sequence by using the chain rule presented in Equation 2.

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t|x_1, x_2, ..., x_{t-1}) \tag{2}$$

A simple strategy to estimate the model in Equation 2 is to compute conditional probability tables for each token given a subset of preceding tokens, according to the classic *n-gram* model. However, this count-based language model technique has well-known limitations, such as high dimensionality and difficulty in dealing with large sequences of n-grams. In this context, the recent BERT-based language model has obtained state-of-the-art results for several natural language processing tasks, in particular, for sequence labeling. Our RE-BERT method can be seen as an extension of the BERT model, i.e. a fine-tuning model for software requirements extraction tasks.

BERT is a contextual neural language model, where the representation of a token is a function of the entire text in which it occurs. Thus, for a given sequence of tokens, we can learn a representation called word embedding $e(x_t)$ for a token x_t, which is also often called a word vector. Semantic proximity between tokens and entire sentences can be computed through their word embeddings, as well as using embeddings as an input to train classifiers in order to obtain a good approximation of the general objective defined in Equation 1. In the context of software requirements extraction, given the token sequence of the review $\mathbf{x} = (x_1, x_2, ..., x_T)$, BERT model first generates a corrupted $\hat{\mathbf{x}}$ version of the sequence, where approximately 15% of the words are randomly selected to be replaced by a special token called [MASK]. Thus, the objective function is to reconstruct the masked tokens $\bar{\mathbf{x}}$ from $\hat{\mathbf{x}}$, according to Equation 3,

$$\max_{\theta} \ log\, p(\bar{\mathbf{x}}|\hat{\mathbf{x}}, \theta) \approx \sum_{t=1}^{T} m_t log \left(\frac{\exp(h_\theta(\hat{\mathbf{x}})_t^\top e(x_t))}{\sum_{x'} \exp(h_\theta(\hat{\mathbf{x}})_t^\top e(x'))} \right) \tag{3}$$

where $e(x_t)$ indicates the embedding of the word x_t; the $h_\theta(\hat{\mathbf{x}})_t$ is a sequence of T hidden state vectors according to parameters θ from the neural network model; and $m_t = 1$ indicates when w_t is masked.

BERT-based models are promising in learning contextual word embeddings from long-term dependencies between tokens in sentences — and even between sentences. However, we note that the extraction of software requirements from reviews is more impacted by a local context, *i.e.* tokens closer to those of software requirements are more important than distant tokens. Thus, motivated by the Local Context Focus mechanism proposed by [16] for sentiment analysis tasks, we investigate local contexts to identify relevant candidates for software requirements. Moreover, our RE-BERT has a training process based on cross-domain learning, where we take advantage of existing labeled data for some apps to learn a model for extracting software requirements in new apps reviews, thereby allowing the use of our pre-trained RE-BERT model as a ready-to-use tool. In the next sections, we detail the fine-tuning process of the proposed RE-BERT.

3.1 Local Context for Software Requirements Extraction

A BERT-based language model can be fine-tuned to find significant correlations between the sequence of tokens in a review $\mathbf{x} = (x_1, x_2, ..., x_T)$ and a sequence of tokens $\mathbf{x}_a = (x_1^a, x_2^a, ..., x_S^a)$ that represents the software requirement, where $\mathbf{x}_a$ is a subsequence of size S (with $S \geq 1$) from $\mathbf{x}$. The global context indicates the model's ability to learn general relationships between the review and the software requirement. Such relationships based on the global context are obtained by BERT's existing next-sentence prediction training strategy. However, we note that global contexts usually fail to correctly identify software requirement tokens, since review tokens that are distant from software requirement tokens can negatively influence the extraction stage. Thus, we investigated the idea of focusing on local contexts to increase the importance of tokens close to the software requirement.

Figure 2 illustrates an app review organized in three parts: (1) software requirement; (2) global context; and (3) local context. Global context tokens capture the behavior and usage scenarios of the app

in relation to the software requirement. For example, in the global context, the "share photos" requirement has the behavior of crashing the app in the usage scenario involving other social networks. The local context is more associated with actions, manners and objects related to the software requirement. Therefore, determining and incorporating the local context during the BERT fine-tuning process is one of the challenges that we believe is important for properly identifying software requirements.

Let $pos(x_i)$ be a function that returns the position of a token x in review $\mathbf{x}$. Equation 4 defines the relative position distance (RD) between a review token and the average position of the software requirement tokens.

$$RD(x_i) = \left| pos(x_i) - \frac{1}{S} \sum_{j=1}^{S} pos(x_j^a) \right| \tag{4}$$

We use a technique to reinforce local context features called Context features Dynamic Weighted (CDW) [16], in which tokens with high RD values will receive a weight decay during BERT's fune-tuning. Let $\mathbf{w} \in \mathbb{R}^h$ be a weight vector used in the attention mechanism of the neural network, where such attention mechanism helps to identify the contextual token embeddings and h is the number of attention layers. We initialize $\mathbf{w}$ with 1 values, indicating that all tokens are of equal importance. During the fine-tuning stage, CDW adjusts the weights according to the RD distance and an α threshold parameter, as defined in Equation 5,

$$\hat{\mathbf{w}}_i = \begin{cases} \mathbf{w}, & \text{if } RD(x_i) \leq \alpha \\ \frac{T \cdot \mathbf{w}}{RD(x_i)+T}, & \text{if } RD(x_i) > \alpha \end{cases} \tag{5}$$

where $\hat{\mathbf{w}}_i$ indicates the weight vector of the token x_i, α is the RD threshold, and T is the size of the token sequence. In the example shown in Figure 2, we use the α threshold equal to 3. Thus, the weight vectors of the gray tokens are preserved, while the white tokens have their weights reduced according to RD values.

3.2 Model Training

In the model training stage, we generate training data from labeled reviews using the BERT structure, $[CLS]\ x_1, x_2, ..., x_T\ [SEP]\ x_i$, for each token x_i of the review. Thus, a review with T tokens will generate T training examples. Each training example is mapped to a label (x_i, y), according to the token x_i, where label y is one of the three labels of the BIO format. The $[CLS]$ and $[SEP]$ tokens are reserved from the BERT model and indicate classification and sentence separation tokens, respectively.

We have two independent representations of a text review. Let $\mathbf{o}^g = BERT^g(\mathbf{x_i})$ be the representation with global context features and $\mathbf{o}^l = BERT^l(\mathbf{x}, \alpha)$ the representation with local context features according to an α parameter. The two representations are concatenated into a single vector $\mathbf{o}^{lg} = [\mathbf{o}^l \oplus \mathbf{o}^g]$ and used as an input data for a dense layer of the neural network, followed by an attention mechanism (MHSA) to obtain the final representation $\hat{\mathbf{z}}$, as defined in Equation 6,

$$\hat{\mathbf{z}} = MHSA(W^{lg} \cdot o^{lg} + b^{lg}) \tag{6}$$

where W^{lg} and b^{lg} are the parameters and bias of the dense layer, respectively, and the MHSA function represents a Multi-Head Self-Attention encoder from BERT's Transformers architecture.

In the output layer, we use the softmax to predict the BIO label of each token, according to Equation 7. Thus, the general sequence labeling task previously described in Equation 1 is approximated when we classify each token in a review. Finally, we use the well-known cross-entropy loss function for training the network.

$$y_l = Softmax(\hat{\mathbf{z}}_l) = \frac{\exp(\hat{\mathbf{z}}_l)}{\sum_{j=1}^{\{B,I,O\}} \exp(\hat{\mathbf{z}}_j)} \tag{7}$$

An important step in RE-BERT training is to use existing labeled review data from different app domains, i.e. a cross-domain learning. This strategy allows the use of RE-BERT to extract software requirements from app domains with no labeled data, since the manual labeling process is an expensive task. Furthermore, we argue that incorporating different domains during the training stage is a robust technique for learning a model with better generalization capabilities.

4 EXPERIMENTAL EVALUATION

We carried out a cross-validation process based on multiple domains to evaluate the proposed RE-BERT. Figure 3 shows this strategy for a scenario involving reviews from 8 different apps. The fine-tuning process of RE-BERT is applied from labeled data with seven different apps. The trained model is then evaluated in the task of extracting software requirements from an eighth app. This process is repeated until all apps are used as a test data.

The following hyperparameters of RE-BERT were defined in the experimental evaluation:

- The learning rate is set to 2×10^{-5};
- The contextual word embedding dimension are set to 768; and
- The α parameter for local context is set to 3.

We compared the RE-BERT performance with three other state-of-the-art methods in the literature: GuMA, SAFE and ReUS, according to the experimental results reported in [4].

4.1 Datasets

We use the datasets created by [4] in the experimental evaluation. According to the authors, the dataset was generated following a rigorous manual annotation process, conducted by two humans, using guidelines from the literature for data selection, labeling, agreement analysis, and reliability analysis.

Table 2 shows some statistics about the datasets. Initially, 341,843 reviews were collected, in free English text, registered by app users in the Play Store and Amazon Store. The reviews are from 8 apps of different categories and time periods. A subset of these reviews, extracted at random, was analyzed manually by the annotators In total, 1,172 different requirements were extracted from 1,000 reviews, which were structured in 2,062 sentences. In the list of labeled software requirements, 530 requirements had only one word and 991 more than one word.

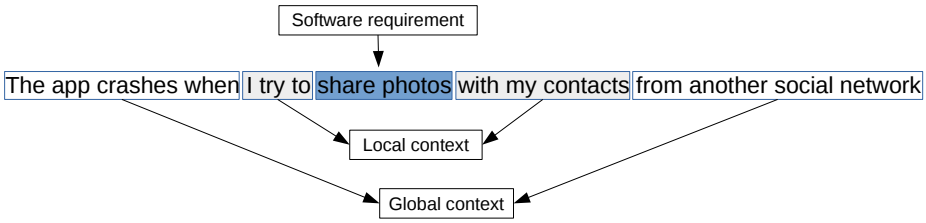

Figure 2: Example of an app review organized in three parts: (1) software requirement; (2) global context; and (3) local context.

Table 1: The overview of the datasets used in the experimental evaluation.

App	reviews	labeled reviews	sentences	features	distinct features	single-word features	multi-word features
Evernote	4,832	125	367	295	259	82	213
Facebook	8,293	125	327	242	204	80	162
eBay	1,962	125	294	206	167	78	128
Netflix	14,310	125	341	262	201	94	168
Spotify	14,487	125	227	180	145	69	111
Photo editor	7,690	125	154	96	80	39	57
Twitter	63,628	125	183	122	99	39	83
WhatsApp	248,641	125	169	118	100	49	69

	App1	App2	App3	App4	App5	App6	App7	App8
Iteration #1	Train	Train	Train	Train	Train	Train	Train	Test
Iteration #2	Train	Train	Train	Train	Train	Train	Test	Train
Iteration #3	Train	Train	Train	Train	Train	Test	Train	Train
Iteration #4	Train	Train	Train	Train	Test	Train	Train	Train
Iteration #5	Train	Train	Train	Test	Train	Train	Train	Train
Iteration #6	Train	Train	Test	Train	Train	Train	Train	Train
Iteration #7	Train	Test	Train	Train	Train	Train	Train	Train
Iteration #8	Test	Train	Train	Train	Train	Train	Train	Train

Figure 3: Cross-domain training strategy involving multiple domain app reviews.

4.2 Evaluation Metrics

We use the F_1 evaluation measure that corresponds to the harmonic mean of Precision (Equation 8) and Recall (Equation 9), where TP (True Positive) refers to the number to features that were both extracted and annotated; FP (False Positive) are features that were extracted but not annotated; and FN (False Negative) refers to the features annotated but not extracted. Equation 10 defines the F_1 measure.

$$P = \frac{TP}{TP + FP}, \quad (8) \qquad R = \frac{TP}{TP + FN} \quad (9) \qquad F_1 = \frac{2 \times P \times R}{P + R}, \quad (10)$$

To determine whether an extracted feature is true or false positive, we compared them with annotated features in the ground truth. To this end, we used the feature matching method[4]: Let Γ be the set of words in a review sentence and $f_i \subseteq \Gamma$ be the set of words used to refer to feature i in that sentence. Two features $f_1, f_2 \subseteq \Gamma$ match at level n (with $n \in \mathbb{N}$) if and only if (i) one of the feature is equal to or is a subset of the other, i.e. $f_1 \subseteq f_2$ or $f_2 \subseteq f_1$, and (ii) the absolute length difference between the features is at most n, i.e. $||f_1| - |f_2|| \leq n$.

4.3 Results and Discussion

We present and discuss the results of software requirements extraction from app reviews considering two scenarios: exact matching and partial matching. In the exact matching scenario, the precision and recall metrics are calculated with matching level $n = 0$, i.e. we consider an error when the model does not identify all the tokens in the software requirement. In the partial matching scenario, we evaluate two levels of matching $n = 1$ and $n = 2$. For example, if the labeled software requirement is "*credit card payment*", then we do not consider an error if the model extracts the "*payment*" token as a requirement, when the matching level is $n = 2$.

Table 2 presents a general comparison of the proposed RE-BERT and three state-of-the-art methods for software requirements extraction. Remember (Section 2) that the GuMa, SAFE and REUS methods are based mainly on linguistic rules from the part-of-speech of the review text, while our RE-BERT is based on a neural language model (contextual word embeddings). The results show that RE-BERT outperforms the three state-of-the-art methods. In particular, RE-BERT performs significantly better in the exact matching scenario.

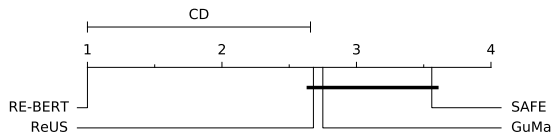

Figure 4: Critical difference diagram for the exact matching scenario.

We carried out a statistical analysis of the experimental results using the Friedman test, followed by the post-hoc Nemenyi. Figure 4 shows a critical difference diagram for the exact matching

Table 2: Comparison (macro F_1 measure) of approaches GuMa,SAFE, ReUS and RE-BERT.

APP	Exact Match (n = 0)				Partial Match 1 (n = 1)				Partial Match 2 (n = 2)			
	GuMa	SAFE	ReUS	RE-BERT	GuMa	SAFE	ReUS	RE-BERT	GuMa	SAFE	ReUS	RE-BERT
Evernote	0.08	0.07	0.07	**0.45**	0.21	0.23	0.19	**0.55**	0.24	0.33	0.28	**0.63**
Facebook	0.04	0.03	0.09	**0.43**	0.15	0.16	0.14	**0.53**	0.19	0.24	0.19	**0.61**
eBay	0.05	0.04	0.06	**0.40**	0.18	0.24	0.14	**0.46**	0.22	0.36	0.21	**0.53**
Netflix	0.05	0.03	0.06	**0.47**	0.18	0.20	0.19	**0.54**	0.21	0.28	0.27	**0.62**
Spotify	0.07	0.04	0.14	**0.44**	0.24	0.23	0.21	**0.51**	0.28	0.35	0.27	**0.60**
PhotoEditor	0.11	0.10	0.13	**0.66**	0.25	0.30	0.22	**0.72**	0.28	0.34	0.26	**0.81**
Twitter	0.09	0.06	0.02	**0.57**	0.24	0.23	0.11	**0.61**	0.27	0.35	0.26	**0.67**
WhatsApp	0.08	0.11	0.06	**0.48**	0.22	0.32	0.19	**0.57**	0.26	0.39	0.24	**0.61**
Mean	0.07	0.05	0.07	**0.46**	0.22	0.23	0.19	**0.55**	0.25	0.34	0.26	**0.62**

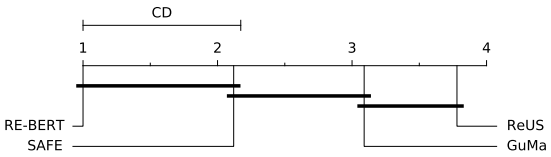

Figure 5: Critical difference diagram for the partial matching scenario.

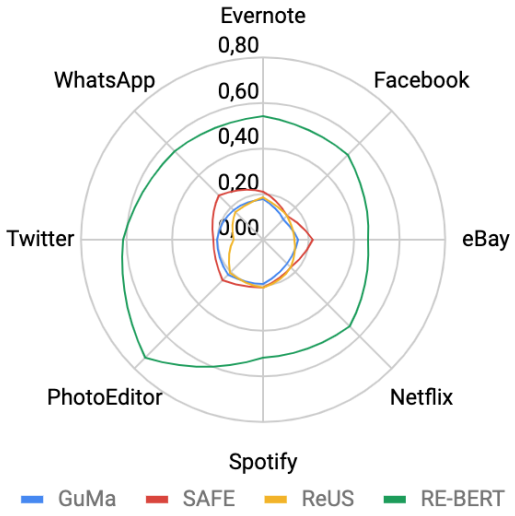

Figure 6: Overall methods comparison using the average F_1 measure calculated from all matching scenarios.

scenario. Each method is ordered according to its average ranking calculated from the F_1 measure of all datasets. The critical difference (CD) indicates a statistically significant difference between two methods. In the diagram, two methods connected by a line indicate no statistically significant difference between them, given the value of critical difference. Thus, in our analysis there is no significant difference between the three methods GuMa, SAFE and ReUS for the exact matching scenario, while RE-BERT presents a statistically superior performance.

Figure 5 presents a similar statistical analysis for the scenario involving partial matching ($n = 1$ and $n = 2$). In this scenario, no significant differences were identified between RE-BERT and SAFE. However, RE-BERT has a statistically superior performance to GuMA and ReUS methods.

Figure 6 shows a general comparison with the average F_1 calculated from all matching scenarios. We emphasize that such promising results are obtained by the fine-tuning step of RE-BERT, where we have a model training phase while the other methods are predefined linguistic rules and heuristics. However, it is worth noting that our RE-BERT model is trained from labeled data of different app reviews domains, which is potentially useful for extracting requirements for new app reviews without labeled data. In this sense, we published a pre-trained RE-BERT model on the GitHub project page *https://omitted-due-to-the-double-blind-review*. This model can be used as a ready-to-use software requirements extractor, just like existing state-of-the-art rule-based models.

We present an example of using the RE-BERT pre-trained in the context of an extractor module (Figure 7). The module receives raw texts from app reviews and returns the classification confidence level of each token belonging to requirement software class *B* (before) or *I* (inside). Thus, in addition to the BIO label, software engineers can explore requirements extracted through different confidence levels.

5 CONCLUSION

App reviews have become an important and valuable knowledge source for requirements engineering. Users often describe details about the app's features they're enjoying or about requirements with bad behavior, as well as compare features with other competing apps. Software requirements extraction tasks from app reviews are the first and crucial step in mining users' opinions for requirements engineering. However, existing literature methods are often based on linguistic rules, which are not suitable for dealing with ambiguities, noise, misspelling, and informal texts. Thus, we introduced RE-BERT, which extends the state-of-the-art neural language models for software requirements extraction tasks. In particular, RE-BERT uses a local context mechanism, thereby allowing to generate word embeddings for tokens according to the sentence's context in which the requirement occurs.

I	use	the	app	to	apply	filters	to	my	photos	and	it's	really	fun	.
0.01	0.25	0.01	0.02	0.87	0.85	0.96	0.87	0.99	0.99	0.01	0.01	0.00	0.00	0.01

Figure 7: Example of using the pre-trained RE-BERT extractor module for app reviews.

Our RE-BERT obtained competitive results and presented a percentage increase of several orders of magnitude in the F_1 measure. In addition, we proposed a training strategy based on cross-domain, which allows the use of RE-BERT to extract requirements from new app reviews without labeled data.

The directions for future work involve extending RE-BERT to the other stages of review mining for requirements engineering. In special, the semantic clustering of the extracted software requirements, in which different writing variations of the same requirement will be standardized according to the formed cluters. Another direction for future work is to incorporate a polarity classification step (positive, negative and neutral) of the extracted requirement, thereby allowing a software requirements-based sentiment analysis.

REFERENCES

[1] Charu Aggarwal. 2018. *Machine Learning for Text.* Springer. https://doi.org/10.1007/978-3-319-73531-3
[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*. 1638–1649.
[3] A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman. 2019. App Store Effects on Software Engineering Practices. *IEEE Transactions on Software Engineering* (2019), 1–1.
[4] Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. 2020. Mining User Opinions to Support Requirement Engineering: An Empirical Study. In *Advanced Information Systems Engineering*, Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant (Eds.). Springer International Publishing, Cham, 401–416.
[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[6] Mauro Dragoni, Marco Federici, and Andi Rexha. 2019. An unsupervised aspect extraction strategy for monitoring real-time reviews stream. *Information Processing and Management* 56, 3 (2019), 1103 – 1118. https://doi.org/10.1016/j.ipm.2018.04.010
[7] E. Guzman and W. Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 153–162.
[8] He Jiang, Jingxuan Zhang, Xiaochen Li, Zhilei Ren, David Lo, Xindong Wu, and Zhongxuan Luo. 2019. Recommending New Features from Mobile App Descriptions. *ACM Trans. Softw. Eng. Methodol.* 28, 4, Article 22 (Oct. 2019), 29 pages. https://doi.org/10.1145/3344158
[9] T. Johann, C. Stanik, A. M. Alizadeh B., and W. Maalej. 2017. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 21–30.
[10] N. Li, L. Zheng, Y. Wang, and B. Wang. 2019. Feature-Specific Named Entity Recognition in Software Development Social Content. In *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*. 175–182.
[11] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. 2016. Toward Data-Driven Requirements Engineering. *IEEE Software* 33, 1 (2016), 48–54.
[12] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. 2017. A Survey of App Store Analysis for Software Engineering. *IEEE Transactions on Software Engineering* 43, 9 (2017), 817–847.
[13] D. Pagano and W. Maalej. 2013. User feedback in the appstore: An empirical study. In *IEEE International Requirements Engineering Conference (RE)*. 125–134. https://doi.org/10.1109/RE.2013.6636712
[14] Mohammadali Tavakoli, Liping Zhao, Atefeh Heydari, and Goran Nenadić. 2018. Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools. *Expert Systems with Applications* 113 (2018), 186 – 199. https://doi.org/10.1016/j.eswa.2018.05.037
[15] Ying Wang, Liwei Zheng, and Ning Li. 2020. ROM: A Requirement Opinions Mining Method Preliminary Try Based on Software Review Data. In *Proceedings of the 2020 4th International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS 2020)*. Association for Computing Machinery, New York, NY, USA, 26–33. https://doi.org/10.1145/3380625.3380665
[16] Biqing Zeng, Heng Yang, Ruyang Xu, Wu Zhou, and Xuli Han. 2019. LCF: A Local Context Focus Mechanism for Aspect-Based Sentiment Classification. *Applied Sciences* 9 (08 2019), 3389. https://doi.org/10.3390/app9163389