Where is Your App Frustrating Users?

Yawen Wang[1,2], Junjie Wang[1,2,3,*], Hongyu Zhang[5], Xuran Ming[1,2], Lin Shi[1,2], Qing Wang[1,2,3,4,*]
[1] Laboratory for Internet Software Technologies, [3] State Key Laboratory of Computer Sciences, [4] Science & Technology on Integrated Infomation System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[5] The University of Newcastle, Callaghan, Australia
{yawen2018, junjie, xuran2020, shilin, wq}@iscas.ac.cn, hongyu.zhang@newcastle.edu.au

ABSTRACT

User reviews of mobile apps provide a communication channel for developers to perceive user satisfaction. Many app features that users have problems with are usually expressed by key phrases such as "upload pictures", which could be buried in the review texts. The lack of fine-grained view about problematic features could obscure the developers' understanding of where the app is frustrating users, and postpone the improvement of the apps. Existing pattern-based approaches to extract target phrases suffer from low accuracy due to insufficient semantic understanding of the reviews, thus can only summarize the high-level topics/aspects of the reviews. This paper proposes a semantic-aware, fine-grained app review analysis approach (SIRA) to extract, cluster, and visualize the problematic features of apps. The main component of SIRA is a novel BERT+Attr-CRF model for fine-grained problematic feature extraction, which combines textual descriptions and review attributes to better model the semantics of reviews and boost the performance of the traditional BERT-CRF model. SIRA also clusters the extracted phrases based on their semantic relations and presents a visualization of the summaries. Our evaluation on 3,426 reviews from six apps confirms the effectiveness of SIRA in problematic feature extraction and clustering. We further conduct an empirical study with SIRA on 318,534 reviews of 18 popular apps to explore its potential application and examine its usefulness in real-world practice.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**; **Software maintenance tools**.

KEYWORDS

App Review, Information Extraction, Deep Learning

ACM Reference Format:
Yawen Wang[1,2], Junjie Wang[1,2,3,*], Hongyu Zhang[5], Xuran Ming[1,2], Lin Shi[1,2], Qing Wang[1,2,3,4,*]. 2022. Where is Your App Frustrating Users?. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510189

* Corresponding author.

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05…$15.00
https://doi.org/10.1145/3510003.3510189

1 INTRODUCTION

Mobile app development has been active for over a decade, generating millions of apps for a wide variety of application domains such as shopping, banking, and social interactions. They have now become indispensable in our daily life. The importance of mobile apps urges the development team to make every endeavor to understand users' concerns and improve app quality.

Users often write reviews of the mobile apps they are using on distribution platforms such as Apple Store and Google Play Store. These reviews are short texts that can provide valuable information to app developers, such as user experience, bug reports, and enhancement requests [16, 27, 41, 52]. A good understanding of these reviews can help developers improve app quality and user satisfaction [15, 30, 48]. However, popular apps may receive a large number of reviews every day. Therefore, manually reading and analyzing each user review to extract useful information is very time-consuming,

Figure 1: An example app review and problematic feature.

In recent years, automated techniques for mining app reviews have attracted much attention [21, 45, 47]. These techniques can help reduce the effort required to understand and analyze app reviews in many ways, such as topic discovery [6, 40, 48], and key phrase extraction [12, 15, 27, 52, 57]. However, existing work about topic discovery can only identify *WHAT* the users complain about [30, 48, 57], such as the high-level topics/aspects of the reviews (e.g., *compatibility*, *update*, *connection*, etc). Taken the review of *Instagram* in Figure 1 as an example, existing approaches would capture terms such as *update, cache, uninstall*, yet missing its core intent. Developers still could not have a concrete understanding about which specific features of the app the users are complaining about. Furthermore, existing work about key phrase extraction mainly utilizes heuristic-based techniques (such as Part-of-Speech patterns,

parsing tree, and semantic dependence graph) to extract the target phrases, which could have insufficient semantic understanding of the reviews. As a result, their accuracy is less satisfactory and can be further improved.

In comparison, we aim at exploiting the *WHERE* aspect of the app reviews, and providing an accurate fine-grained landscape about *where an app frustrates the users*, i.e., which specific app features[1] the users have problems with. As an example in Figure 1, the review is about a crashing problem, and the problematic feature the user complained about is *upload to my story*. The fine-grained knowledge about problematic features could facilitate app developers in understanding the user concerns, localizing the problematic modules, and conducting follow-up problem-solving activities.

To overcome the drawbacks of existing work and better exploit the app reviews, this paper proposes a Semantic-aware, fIne-grained app Review Analysis approach (SIRA), which can extract, cluster, and visualize the problematic features of apps. More specifically, SIRA includes a novel BERT+Attr-CRF model to automatically extract the fine-grained phrases (i.e., problematic features). It combines the review descriptions and review attributes (i.e., app category and review description sentiment) to better model the semantics of reviews and boost the performance of the traditional BERT-CRF model [63]. With the extracted phrases, SIRA then designs a graph-based clustering method to summarize the common aspects of problematic features based on their semantic relations. Finally, SIRA presents a visualization of the summarized problematic features.

We evaluate SIRA on 3,426 reviews involving 8,788 textual sentences from six apps spanning three categories. For problematic feature extraction, the overall precision and recall achieved by SIRA is 84.27% and 85.06% respectively, significantly outperforming the state-of-the-art methods. SIRA can also achieve high performance in problematic feature clustering, outperforming two commonly-used baselines. We further conduct an empirical study with SIRA on 318,534 reviews of 18 popular apps (reviews spanning 10 months) to explore its potential application and examine its usefulness in real-world practice. We find that different apps have their unique problematic features and problematic feature distributions. The results also reveal that different apps can share some common problematic features. This observation can facilitate mobile app testing, e.g., recommending bug-prone features to similar apps for test prioritization.

The main contributions of this paper are as follows:

- A semantic-aware, fine-grained app review analysis approach (SIRA) to extracting, clustering, and visualizing the problematic features of apps. In SIRA, we design a BERT+Attr-CRF model to automatically extract the fine-grained phrases (i.e., problematic features), and a graph-based clustering method to summarize the common aspects of problematic features.
- The evaluation of the proposed SIRA on 3,426 reviews involving 8,788 textual sentences from six apps spanning three categories, with affirmative results.
- A large-scale empirical study on 318,534 reviews of 18 popular apps, to explore its potential application and usefulness in real-world practice.
- Public accessible source code and experimental data at https://github.com/MeloFancy/SIRA.

[1]We refer to a feature as a distinctive, user-visible characteristic of a mobile app [29][65], e.g., sending videos, viewing messages, etc.

2 BACKGROUND AND RELATED WORK

Named Entity Recognition (NER). NER is a classic Natural Language Processing (NLP) task of sequence tagging [25, 66]. Given a sequence of words, NER aims to predict whether a word belongs to named entities, e.g., names of people, organizations, locations, etc. NER task can be solved by linear statistical models, e.g., Maximum Entropy Markov models [43, 53], Hidden Markov Models [11] and Conditional Random Fields (CRF) [34]. Deep learning-based techniques would use a deep neural network to capture sentence semantics and a CRF layer to learn sentence-level tag rules. Typical network structures include convolutional neural network with CRF (Conv-CRF) [7], Long Short-Term Memory network with CRF (LSTM-CRF) and bidirectional LSTM network with CRF (BiLSTM-CRF) [25]. By taking advantage of the bidirectional structure, BiLSTM-CRF model can use the past and future input information and can usually obtain better performance than Conv-CRF and LSTM-CRF.

Language model pre-training techniques have been shown to be effective for improving many NLP tasks [10, 22]. BERT (Bidirectional Encoder Representations from Transformers) [10] is a Transformer-based [55] representation model that uses pre-training to learn from the raw corpus, and fine-tuning on downstream tasks such as the NER task. Employing BERT to replace BiLSTM (short for BERT-CRF) could lead to further performance boosts [63]. BERT-CRF model benefits from the pre-trained representations on large general corpora combined with fine-tuning techniques.

Mining user reviews. Harman et al. introduced the concept of app store mining by identifying correlations between the customer ratings and the download rank of a mobile app [21, 42]. Palomba et al. found that developers implementing user reviews would be rewarded in terms of app ratings [47]. Noei et al. investigated the evolution of app ranks and identified the variables that share a strong relationship with ranks, e.g., number of releases [45].

Previous studies on mining user reviews emphasized the topic discovery/classification and summarization of reviews as a way of aggregating a large amount of text and reducing the effort required for analysis [6, 40, 46, 48, 52]. These classifications are from different points of view, e.g., whether or not the reviews include bug information, requests for new features [40], whether they are informative [6], whether reviews across different languages and platforms are similar [46], or based on a taxonomy relevant to software maintenance and evolution [48], etc. Other studies focused on the information extraction from app reviews considering the fact that reading through the entire reviews is impractical [12, 15, 16, 30, 33, 57]. For example, the types of complains [30], the app aspects loved by users [15], user rationale [33] and summaries for guiding release planning [56] are extracted and summarized for facilitating the review understanding.

There are some studies on mining API-related opinions from informal discussions, such as Q&A websites (e.g., Stack Overflow)

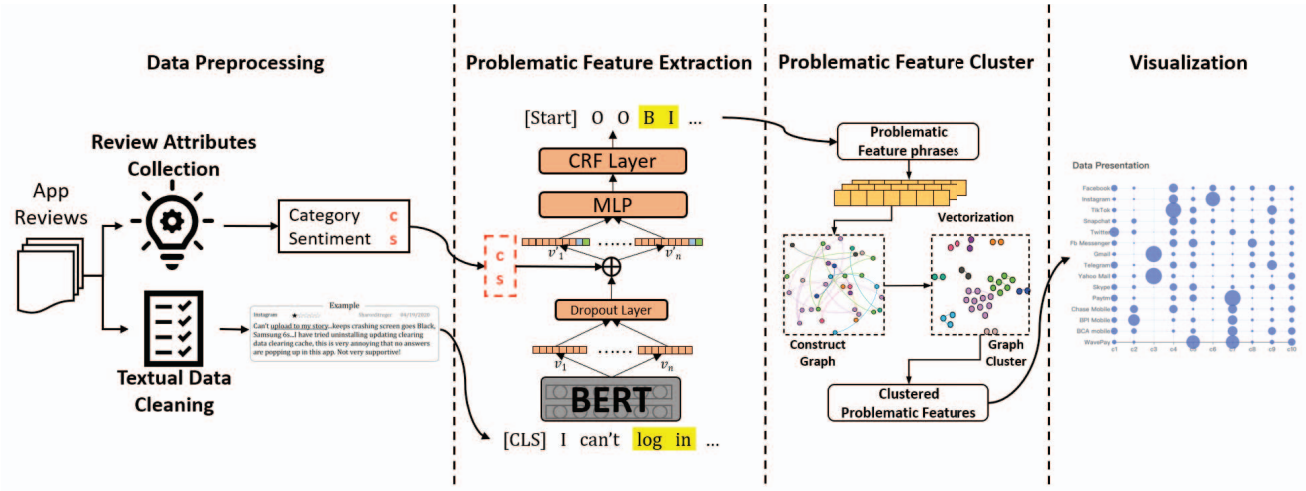


Figure 2: The overview of SIRA.

to alleviate developers' burden in performing manual searches [38, 54]. These methods mainly depend on fuzzy matching with pre-built API databases, which cannot work in our context. There are also some studies on mining social media data (e.g., Twitter data) [18]. The app reviews mainly convey users' feedback about an app, while the Twitter data is more general and contains daily messages. Therefore, general-purpose techniques for Twitter data require customizations to better understand app reviews.

Some studies are similar to our work, such as topic discovery/classification, sentiment analysis, etc. However, they do not support the extraction of fine-grained features well. For example, INFAR [12] mines insights from app reviews and generates summarizes after classifying sentences into pre-defined topics. The discovered topics from INFAR are more coarse-grained (e.g., GUI, crash, etc.). Our method can highlight the fine-grained features (e.g., "push notification") that users complained about; SUR-Miner [15] and Caspar [16] uses techniques, such as dependency parsing and Part-of-Speech pattern, to extract some aspects from app reviews. Guzman et al. [19] proposed a method, which can only extract features consisting of two words (i.e., collocations) from the reviews based on word co-occurrence patterns, which is not applicable in our context, because the problematic features might contain multiple words; Opiner [54] is a method to mining aspects from API reviews. It extracts API mentions from API reviews through exact and fuzzy name matching with pre-built API databases, which is difficult to work in our context because we do not have a database of feature phrases in advance. These studies utilized pattern-based method to extract the target phrases, which did not consider the review semantics sufficiently, and had bad tolerance to noise; by comparison, our proposed approach is a semantic-aware approach.

Mining open source bug reports. Previous studies have proposed various methods to automatically classify bug reports [28, 39], detect the duplicate reports [8, 60, 67], summarize the reports [20], and triage the reports [23, 36, 62], etc. The bug reports in open source or crowd testing environment are often submitted by software practitioners, and often described with detailed bug explanation and in relatively longer length. Yet the app reviews are submitted by the end users and in much fewer words, thus the above mentioned approaches could not be easily adopted in this context.

Semantic-aware approaches in SE. Researchers have utilized deep learning based techniques to capture the semantics of software artifacts and facilitate the follow-up software engineering tasks. Such kinds of studies include neural source code summarization with attentional encoder-decoder model based on code snippets and summaries [64], requirement traceability by incorporating requirements artifact semantics and domain knowledge into the tracing solutions [17], knowledge mining of informal discussions on social platforms [59], etc. This paper focuses on a different type of software artifact (i.e., app reviews) and incorporates a state-of-the-art technique (i.e., BERT) for the semantic-aware learning, and the results show its effectiveness.

3 APPROACH

This paper proposes a Semantic-aware, fIne-grained app Review Analysis approach SIRA to extract, cluster, and visualize the problematic features of apps (i.e., the phrases in app reviews depicting the feature which users have problems with, see the examples in Figure 1.)

Figure 2 presents the overview of SIRA, which consists of four steps. ***First***, it preprocesses the app reviews crawled from online app marketplace, to obtain the cleaned review descriptions and the review attributes (i.e., the category of the belonged app c and the review description sentiment s). ***Second***, it builds and trains a BERT+Attr-CRF model to automatically extract the fine-grained phrases about problematic features. BERT+Attr-CRF combines the review descriptions and two review attributes as input to better model the semantics of reviews and boost the phrase extraction performance of the traditional BERT-CRF model. ***Third***, SIRA clusters the extracted phrases with a graph-based clutering method to

summarize the common aspects of problematic features based on their semantic relations. And ***finally***, it presents a visualization view to illustrate the summaries and compare the problematic features among apps, in order to acquire a better understanding of where users complain about across apps.

3.1 Data Preprocessing

Data preprocessing mainly includes two steps: textual data cleaning and review attribute collection.

3.1.1 *Textual Data Cleaning*.

The raw app reviews are often submitted via mobile devices and typed using limited keyboards. This situation leads to the frequent occurrences of massive noisy words, such as repetitive words, misspelled words, acronyms and abbreviations [13, 15, 57, 58].

Following other CRF-based practices [25], we treat each sentence as an input unit. We first split each review into sentences by matching punctuations through regular expressions. Then we filter all non-English sentences with Langid[2]. We tackle the noisy words problem with the following steps:

- **Lowercase**: we convert all the words in the review descriptions into lowercase.
- **Lemmatization**: we perform lemmatization with Spacy[3] to alleviate the influence of word morphology.
- **Formatting**: we replace all numbers with a special symbol "<number>" to help the BERT model unify its understanding. Besides, we build a list containing all the app names crawled from Google Play Store, and replace them with a uniform special symbol "<appname>".

3.1.2 *Review Attribute Collection*.

Some attributes related to the review or the app can facilitate the extraction of problematic features in Section 3.2. This subsection collects these attributes, i.e., the category of the belonged app c and the review description sentiment s as shown in Figure 2 and Figure 3. The reason why we include the app category is that apps from different categories would exert unique nature in terms of functionalities and topics [14]. Furthermore, review descriptions with negative sentiment would be more likely to contain problematic features, compared with the description with positive sentiment. Hence, we include review description sentiment as the second attribute in our model.

App categories can be directly collected when crawling data from Google Play Store. To obtain the sentiment for each review sentence, we employ SentiStrength-SE [26], a domain-specific sentiment analysis tool especially designed for software engineering text. SentiStrength-SE would assign a positive integer score in the range of 1 (not positive) to 5 (extremely positive) and a negative integer score in the range of -1 (not negative) to -5 (extremely negative) to each sentence. Employing two scores is because previous research from psychology [2] has revealed that human beings process the positive and negative sentiment in parallel. Following previous work [14, 19], if the absolute value of the negative score multiplied by 1.5 is larger than the positive score, we assign the sentence the negative sentiment score; otherwise, the sentence is assigned with the positive sentiment score.

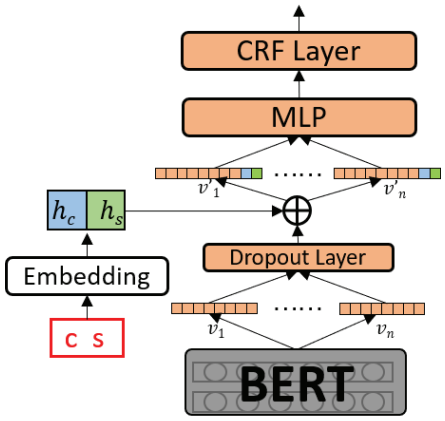


Figure 3: Detailed structure of BERT+Attr-CRF.

3.2 Problematic Feature Extraction

We model the problematic feature extraction problem as a Named Entity Recognition (NER) task, where we treat problematic features as named entities, and solve the problem with the commonly-used CRF technique. To better capture the semantics of the app reviews, we employ the BERT model to encode the review descriptions. Furthermore, we incorporate the review attributes in the CRF model to further boost the recognition of problematic features. Two attributes, i.e., category of the belonged app c and review description sentiment s (see Section 3.1.2), are utilized in our model.

Following other NER tasks, we use the BIO tag format [9, 50] to tag each review sentence, where

- B-label (Beginning): The word is the beginning of the target phrase.
- I-label (Inside): The word is inside the target phrase but not its beginning.
- O-label (Outside): The word is outside the target phrase.

The BIO-tagged review sentence is input into the BERT+Attr-CRF model for further processing.

Figure 3 presents the detailed structure of our proposed BERT+Attr-CRF model. Since app reviews are short texts, and the involved vocabulary is relatively small, we use the pre-trained model $BERT_{BASE}$[4], which has 12 layers, 768 hidden dimensions and 12 attention heads. It has been pre-trained on the BooksCorpus (800M words) and English Wikipedia (2,500M words), and will be fine-tuned using our own data. Each input sentence is represented by 128 word tokens with a special starting symbol $[CLS]$. For those not long enough, we use a special symbol $[PAD]$ to pad them to the length of 128, following the common practice. The outputs of BERT are fed into a dropout layer to avoid over-fitting. Finally, we obtain n (the length of the input sentence) vectors, with each vector (denoted as v_i) having 768 dimensions and corresponding to each input word.

[2]https://github.com/saffsd/langid.py
[3]https://spacy.io
[4]https://huggingface.co/bert-base-uncased

We incorporate the review attributes into the textual vectors (v) to jointly capture the underlying meaning of the review sentence. The review attributes (c and s) extracted in Section 3.1.2 are discrete values. We first convert them into continuous vectors (denoted as h_c and h_s) by feeding them into the embedding layers. Taking attribute s as an example, it can take ten values (-5 to -1 and 1 to 5). The embedding layer could represent each value with a continuous vector, which can be trained jointly with the whole model. We then concatenate h_c, h_s and v ($h_c \bigoplus h_s \bigoplus v$) to obtain a vector (denoted as v_i') for each input word. The concatenated vectors first go through a Multi-layer Perceptron (MLP), which computes the probability vector (denoted p) of BIO tags for each word:

$$p = f(W[h_c; h_s; v]) \tag{1}$$

where $f(\cdot)$ is the activation function, and W is trainable parameters in MLP. $[h_c; h_s; v]$ is the concatenation of these three vectors. Finally, p is input into the CRF layer to determine the most likely tag sequence based on Viterbi Algorithm [1].

Based on the derived tag sequence, we can obtain the phrases about problematic features. For example, if our input review sentence is "whenever I go to send a video it freezes up", and the output tag sequence is "$< O >< O >< O >< O >< B >< I >< I >< O >< O >< O >$", we can determine the extracted problematic feature as "send a video" based on the BIO format.

The loss function of the model should measure the likelihood of the whole true tag sequence, instead of the likelihood of the true tag for each word in the sequence. Therefore, the commonly-used Cross Entropy is not suitable in this context. Following existing studies [25], the loss function contains two parts: the emission score and the transition score. It is computed as:

$$s([x]_1^T, [l]_1^T, \widetilde{\theta}) = \sum_{t=1}^{T} ([A]_{[l]_{t-1},[l]_t} + [f_\theta]_{[l]_t,t}) \tag{2}$$

where $[x]_1^T$ is the sentence sequence of length T, and $[l]_1^T$ is the tag sequence. $f_\theta([x]_1^T)$ is the emission score, which is the output of MLP with parameters θ, and $[A]_{i,j}$ is the transition score, which is obtained with the parameters from the CRF layer. The transition score $[A]_{i,j}$ models the transition from the i-th state to the j-th state in the CRF layer. $\widetilde{\theta} = \theta \cup \{[A]_{i,j} \forall i, j\}$ is the new parameters for the whole network. The loss of a sentence $[x]_1^T$ along with a sequence of tags $[l]_1^T$ is derived by the sum of emission scores and transition scores.

Model Training: The hyper-parameters in SIRA are tuned carefully with a greedy strategy to obtain the best performance. Given a hyper-parameter P and its candidate values $\{v_1, v_2, ..., v_n\}$, we perform automated tuning for n iterations, and choose the values which leads to the best performance as the tuned value of P. After tuning, the learning rate is set as 10^{-4}. The optimizer is Adam algorithm [31]. We use the mini-batch technique for speeding up the training process with batch size 32. The drop rate is 0.1, which means 10% of neuron cells will be randomly masked to avoid over-fitting.

We implement this BERT+Attr-CRF model using Transformers[5], which is an open-source Pytorch library for Natural Language Understanding and Natural Language Generation. Our implementation and experimental data are available online[6].

3.3 Problematic Feature Clustering

The extracted problematic features might be linguistically different yet semantically similar. To provide a summarized view of the problematic features, this step clusters the extracted problematic features based on the topics derived from their semantic relations. Conventional topic models use statistical techniques (e.g., Gibbs sampling) based on word co-occurrence patterns [49]. They are not suitable for the short texts (i.e., problematic features in our context), because the co-occurrence patterns can hardly be captured from the short text, instead the semantic information should be taken into consideration. Additionally, these models need to specify the number of clusters/topics, which is hardly determined in our context. To tackle these challenges, we design a graph-based clustering method, which employs semantic relations of problematic features.

First, we convert problematic feature phrases into 512 dimensional vectors using Universal Sentence Encoder (USE) [5]. It is a transformer-based sentence embedding model that captures rich semantic information, and has been proven more effective than traditionally-used word embedding models [16]. **Second**, we construct a weighted, undirected graph, where each problematic feature is taken as a node, and the cosine similarity score between USE vectors of two problematic features is taken as the weight between the nodes. If the score is over a certain ratio, we add an edge between two nodes. The ratio is an input hyper-parameter, which measures the semantic correlations between problematic features. The higher ratio leads to higher cluster cohesion. We set it as 0.5 after tuning in the training data. **Third**, we perform Chinese Whispers (CW) [3], which is an efficient graph clustering algorithm, on this graph to cluster problematic features.

With this graph-based clustering method, SIRA can group the problematic features that are semantically similar into the same topic. We implement our clustering method in python, based on the open-source implementation of USE[7] and CW[8].

3.4 Visualization

In order to display the clustering results of multiple apps more intuitively, we provide a visualized view in the form of bubble charts (an example is shown in Figure 4). The y-axis demonstrates the names of investigated apps, and the x-axis represents the id of each cluster. The size of the bubble (denoted as $s_{a,c}$) of app a in cluster c is defined as the ratio between the number of problematic features of app a in cluster c and the total number of problematic features in app a.

When the cursor hovers over the bubble, it would display detailed information of this cluster, including the cluster name, the number of problematic features, and example reviews with corresponding problematic features. For the cluster name, we first find the most

[5]https://github.com/huggingface/transformers
[6]https://github.com/MeloFancy/SIRA
[7]https://github.com/MartinoMensio/spacy-universal-sentence-encoder
[8]https://github.com/nlpub/chinese-whispers-python

Table 1: Experimental dataset.

Category	App	# Reviews	# Sentences
Social	Instagram	582	1,402
	Snapchat	585	1,388
Communication	Gmail	586	1,525
	Yahoo Mail	542	1,511
Finance	BPI Mobile	588	1,488
	Chase Mobile	543	1,474
Overall		3,426	8,788

frequent noun or verb (denoted as w) among all problematic features in the cluster. We then count the number of problematic features containing w, and treat the most frequent phrase as the cluster name (i.e., the representative problematic feature). By comparing the relative sizes of bubbles, one can intuitively acquire the distribution of problematic features across apps.

4 EXPERIMENTAL DESIGN

4.1 Research Questions

We answer the following three research questions:

- **RQ1:** What is the performance of SIRA in extracting problematic features?
- **RQ2:** Is each type of the review attributes employed in SIRA necessary?
- **RQ3:** What is the performance of SIRA in clustering problematic features?

RQ1 investigates the performance of SIRA in problematic feature extraction, and we also compare the performance with four state-of-the-art baselines (see Section 4.3) to further demonstrate its advantage. **RQ2** conducts comparison with SIRA's three variants to demonstrate the necessity of the employed review attributes in BERT+Attr-CRF model. **RQ3** investigates the performance of SIRA in problematic feature clustering, and we also compare SIRA with two commonly-used baselines (see Section 4.3).

4.2 Data Preparation

We use the reviews of six apps from three categories (two in each category) in our experiments. All six apps are popular and widely-used by a large number of users. We first crawl the app reviews from Google Play Store submitted during August 2019 to January 2020, with the tool google-play-scraper[9]. For each app, we then randomly sample around 550 reviews (about 1500 sentences) and label them for further experiments. Table 1 elaborates the statistics of the experimental dataset in detail. It contains 3,426 reviews and 8,788 sentences in total.

Three authors then manually label the app reviews to serve as the ground-truth in verifying the performance of SIRA. To guarantee the accuracy of the labeling outcomes, the first two authors firstly label the app reviews of an app independently, i.e., mark the beginning and ending position of the problematic features in each review sentence. Second, the fourth author compares the labeling results, finds the difference, and organizes a face-to-face discussion among them three to determine the final label. All the six apps follow the same process. For the first labeled app (*Instagram*), the Cohen's Kappa is 0.78 between the two participants, while for the last labeled app (*Chase Mobile*), the Cohen's Kappa is 0.86. After two rounds of labeling, a common consensus is reached for every review sentence.

[9]https://github.com/facundoolano/google-play-scraper

4.3 Baselines

4.3.1 *Baselines for Problematic Feature Extraction.*

We select methods that can extract target phrases from app reviews as baselines for problematic feature extraction. To the best of our knowledge, existing methods are mainly pattern-based, which can be classified into three types based on the techniques: 1) Part-of-Speech (PoS) Pattern: SAFE [27] and PUMA [58]; 2) Dependency Parsing plus PoS Pattern: Caspar [16] and SUR-Miner [15]; 3) Pattern-based Filter plus Text Classification: KEFE [61]. We select the representative method from each type as baselines, i.e., KEFE, Caspar, and SAFE. In addition, since we model the feature extraction as an NER task, we also include BiLSTM-CRF [25], a commonly-used technique in NER tasks, as a baseline. We introduce four baselines in detail below:

BiLSTM-CRF [25]: A commonly-used algorithm in sequence tagging tasks such as NER. Being a deep learning-based technique, it utilizes a BiLSTM to capture sentence semantics and a CRF layer to learn sentence-level tags.

KEFE [61]: A state-of-the-art approach for identifying key features from app reviews. A key feature is referred as the features that are highly correlated to app ratings. It firstly employs a pattern-based filter to obtain candidate phrases, and then a BERT-based classifier to identify the features. Since its patterns are designed for Chinese language, we replace them with the patterns in SAFE [27] to handle English reviews.

Caspar [16]: A method for extracting and synthesizing user-reported mini stories regarding app problems from reviews. We treat its first step, i.e., events extraction, as a baseline. An event is referred as a phrase that is rooted in a verb and includes other attributes related to the verb. It employed pattern-based and grammatical NLP techniques such as PoS tagging and dependency parsing on review sentences to address this task. We use the implementation provided by the original paper[10].

SAFE [27]: A method for extracting feature-related phrases from reviews by 18 PoS patterns. For example, the pattern *Verb-Adjective-Noun* can extract features like "delete old emails". We implement all 18 patterns to extract the phrases based on the NLP toolkit NLTK[11].

4.3.2 *Baselines for problematic feature Clustering.*

We employ the following two baselines for problematic feature clustering, which are commonly used for mining topics of app reviews:

K-Means: It is a commonly-used clustering algorithm, and was employed to cluster the keywords of app reviews [57]. In this work, we first encode each problematic feature with TF-IDF [51] vectors, then run K-Means to cluster all problematic features into topics, following previous work [57]. We apply the implementation in the library scikit-learn[12].

[10]https://hguo5.github.io/Caspar
[11]https://github.com/nltk/nltk
[12]https://scikit-learn.org

LDA [4]: It is a commonly-used topic clustering algorithm, and was utilized to group the app features [19]. In this work, we treat the extracted problematic features as documents and run LDA for topic modeling, following previous work [19]. We employ the implementation in the library Gensim[13].

4.4 Experimental Setup

To answer RQ1, we conduct nested cross-validation [32] on the experimental dataset. The inner loop is for selecting optimal hyper-parameters, which are used for evaluating performance in the outer loop. In the outer loop, we randomly divide the dataset into ten folds, use nine of them for training, and utilize the remaining one fold for testing the performance. The process is repeated for ten times, and the average performance is treated as the final performance. In the inner loop, we use eight folds for training and one fold for validation. We run each baseline (see Section 4.3) to obtain its performance following the same experimental setup, and present the evaluation results on each app and on the overall dataset, respectively.

For RQ2, we design three variants of BERT+Attr-CRF model to demonstrate the necessity of employed review attributes in our model architecture. In detail, BERT-CRF, BERT+Cat-CRF, and BERT+SEN-CRF respectively represent the model without review attributes (i.e., only with text), the model without review description sentiment (i.e., with text and app category), and the model without app category (i.e., with text and review description sentiment). We reuse other experimental setups as RQ1.

For RQ3, we manually build the ground-truth clustering results to evaluate the problematic feature clustering performance. The criteria for labeling are to group the features that represent the same functionality into one cluster. More specifically, we randomly sample 100 problematic features for each app (600 in total) derived from the results of RQ1. The two authors independently label these problematic features into clusters in the first round, where the Cohen's Kappa between two authors reaches 0.81 (i.e., a satisfactory degree of agreement). Then follow-up discussions are conducted until common consensus is reached. Finally, the 600 problematic features were labeled into 20 groups. Note that we do not specify the number of clusters in advance, because it is hard to decide the number in our context. Our proposed clustering method does not need to specify this parameter as well. Meanwhile, we run our approach and each baseline (see Section 4.3) to cluster these problematic features, and obtain each approach's clustering performance by comparing the predicted and ground-truth clustering results for each app and the overall dataset, respectively.

The experimental environment is a desktop computer equipped with an NVIDIA GeForce RTX 2060 GPU, intel core i7 CPU, 16GB RAM, running on Windows 10, and training the model takes about 2.5 hours for each fold nested cross-validation.

4.5 Evaluation Metrics

4.5.1 *Metrics for Problematic Feature Extraction*.

We use precision, recall, and F1-Score, which are commonly-used metrics, to evaluate the performance of SIRA for problematic feature extraction. We treat a problematic feature is correctly predicted if the predicted phrase from SIRA for a review sentence of an app is the same as the ground-truth one. Three metrics are computed as:

- **Precision** is the ratio of the number of correctly predicted phrases to the total number of predicted phrases.
- **Recall** is the ratio of the number of correctly predicted phrases to the total number of ground-truth phrases.
- **F1-Score** is the harmonic mean of precision and recall.

4.5.2 *Metrics for Problematic Feature Clustering*.

Following previous work [24], we use the commonly-used Adjusted Rand Index (ARI) [35] and Normalized Mutual Information (NMI) [44] to evaluate the clustering performance by comparing with the ground-truth clustering results. Higher metric values indicate better clustering performance. For clarity, we denote G as the ground-truth clustering result, and C as the predicted clustering result.

Adjusted Rand Index (ARI): It takes values in $[-1, 1]$, reflecting the degree of overlap between the two clusters. The raw Rand Index (RI) is computed by $RI = \frac{a+b}{\binom{n}{2}}$, where a is the number of pairs that are assigned in the same cluster in G and also in the same cluster in C, and b is the number of pairs that are assigned in different clusters both in G and C. $\binom{n}{2}$ is the total number of unordered pairs in a set of n phrases. The raw RI score is then "adjusted for chance" into the ARI score using the following scheme:

$$ARI = \frac{RI - E(RI)}{max(RI) - E(RI)} \tag{3}$$

where $E(RI)$ is the expected value of RI. In this way, the ARI can be ensured to have a value close to 0.0 for random labeling independently of the number of clusters and samples.

Normalized Mutual information (NMI): It measures the similarity degree of the two sets of clustering results between 0 (no mutual information) and 1 (perfect correlation).

$$NMI(G, C) = \frac{MI(G, C)}{\sqrt{H(G)H(C)}} \tag{4}$$

where $H(G) = -\sum_{i=1}^{|G|} P(i) log(P(i))$ is the entropy of set G, and $P(i) = \frac{G_i}{N}$ is the probability that a phrase picked randomly falls into cluster G_i. The $MI(G, C)$ is the mutual information of G and C, i.e., $MI(G, C) = \sum_{i=1}^{|G|} \sum_{j=1}^{|C|} P(i, j) log\left(\frac{P(i,j)}{P(i)P(j)}\right)$.

5 RESULTS AND ANALYSIS

5.1 Answering RQ1

The last column of Table 2 presents the performance of SIRA in problematic feature extraction. The overall precision, recall and F1 are 84.27%, 85.06% and 84.64% respectively, which indicates that 84.27% of problematic features extracted by SIRA are correct, and 85.06% problematic features are correctly extracted from the ground-truth ones. The results confirm that our proposed approach can accurately extract the problematic features.

More specifically, SIRA reaches the highest precision of 90.27% on *Gmail* and the highest recall of 87.37% on *Yahoo Mail*. Its lowest precision is 79.18% on *Yahoo Mail* and the lowest recall is 84.15% on *Snapchat*. We can see that even with its worst performance, an acceptable precision and recall can be achieved.

We then examine the extracted problematic features in detail, and find that there are indeed some observable patterns associated

[13] https://radimrehurek.com/gensim

Table 2: Evaluation on problematic feature extraction (RQ1).

App \ Metric \ Method		KEFE	Caspar	SAFE	BiLSTM-CRF	SIRA
Instagram	**P**	40.32%	16.26%	14.17%	80.24%	**83.59%**
	R	60.76%	10.49%	70.61%	71.79%	**85.70%**
	F1	48.29%	12.46%	23.55%	75.58%	**84.53%**
Snapchat	**P**	42.08%	18.87%	12.95%	78.49%	**82.63%**
	R	58.71%	13.81%	65.60%	74.71%	**84.15%**
	F1	48.70%	15.74%	21.59%	76.47%	**83.30%**
Gmail	**P**	53.79%	25.60%	22.25%	87.58%	**90.27%**
	R	78.54%	9.88%	**88.21%**	71.74%	84.16%
	F1	63.46%	14.12%	35.49%	78.81%	**87.09%**
Yahoo Mail	**P**	12.57%	18.26%	12.57%	74.45%	**79.18%**
	R	70.10%	11.85%	70.10%	74.69%	**87.37%**
	F1	21.25%	14.19%	21.25%	74.26%	**83.00%**
BPI Mobile	**P**	41.92%	20.98%	18.22%	82.58%	**87.37%**
	R	62.75%	9.24%	77.05%	73.53%	**85.07%**
	F1	50.13%	12.51%	29.44%	77.63%	**86.13%**
Chase Mobile	**P**	36.98%	17.53%	12.17%	77.23%	**80.32%**
	R	52.85%	13.38%	64.85%	68.43%	**84.59%**
	F1	43.16%	15.03%	20.44%	72.31%	**82.26%**
Overall	**P**	42.79%*	19.14%*	15.51%*	80.40%	**84.27%**
	R	63.50%*	11.27%*	73.94%**	72.48%*	**85.06%**
	F1	51.05%*	14.13%*	25.62%*	76.15%*	**84.64%**

Compared to SIRA, statistical significance $p - value < 0.05$ is denoted by **, and $p - value < 0.01$ is denoted by *.

with the problematic features. For example, users would use some negative words (e.g., "cannot", "hardly") or temporal conjunctions (e.g., "as soon as", "when") before mentioning the problematic features. This could probably explain why the pattern-based technique [12, 16, 27] could work sometimes. Taking the review in Figure 1 as an example, extracting the phrases after the negative word "can't" would obtain the correct phrase. However, the pattern-based techniques highly rely on the manually defined patterns and have poor scalability in a different dataset. Furthermore, there are many circumstances when the pattern-based approach can hardly work. For example, it is quite demanding to design patterns for the following review sentence: "this update takes away my ability to view transactions", where the problematic feature is "view transaction". These circumstances further prove the advantages and flexibility of our approach.

We also examine the bad cases where SIRA fails to work. In some cases, SIRA can extract the core nouns and verbs of the target phrase, but misses or additionally extracts some trivial words, especially some adverbs/adverbials before or after the core phrase. For example, SIRA might wrongly extract "received emails for 10 days" from "I have not received emails for 10 days", where the ground-truth phrase is "received emails". Such results pull down the performance. This could be improved by considering PoS patterns of words when vectorizing review sentences in future work.

Comparison with baselines. Table 2 presents the performance of SIRA and four baselines in extracting problematic features. SIRA outperforms all baselines on all metrics. This indicates that these pattern-based baselines (i.e., KEFE, Caspar and SAFE) are far from effective in extracting problematic features, while the deep learning-based baseline (i.e., BiLSTM-CRF) is a bit worse than SIRA because of the inferior semantic understanding and neglect of review attributes. To further intuitively demonstrate the advantages of SIRA, Table 3 presents two example reviews and the corresponding problematic features extracted by SIRA and four baselines.

Among the three pattern-based baselines, SAFE achieves 15.51% precision and 73.94% recall. This is because it defines 18 PoS patterns for feature-related phrases, and can retrieve a large number of possible problematic features (i.e., high recall). For example, in the first example of Table 3, SAFE would return two phrases. By comparison, Caspar only extracts events from reviews containing temporal conjunctions and key phrases, including "when", "every time", which can hardly work well in this context. Taking the first review in Table 3 as an example, Caspar can only extract the two phrases/clauses. KEFE achieves the promising performance, indicating that it can filter away many low-quality phrases with the BERT classifier; yet the classification is still conducted based on candidate phrases extracted by a pattern-based method, which limits its performance. In the first example of Table 3, KEFE can filter the wrong phrase "keeps crashing", but the reserved phrase "take a picture" is still not accurate enough due to the drawback of pattern-based candidate phrases. BiLSTM-CRF can achieve promising performance but still not as accurate as our proposed SIRA, e.g., "view story" in Table 3. SIRA can be regarded as an improved version of BiLSTM-CRF, which employs BERT fine-tuning technique and two customized review attributes. The features extracted by SIRA is the superset of BiLSTM-CRF, which can be also reflected by the results in Table 2. SIRA outperforms BiLSTM-CRF in both recall and precision, indicating that SIRA can extract features more accurately and retrieve more problematic features.

5.2 Answering RQ2

Table 4 presents the performance of SIRA and its three variants, respectively. The overall performance of SIRA is higher than all the three variants. Compared with the base BERT-CRF model, adding the app category and the sentiment attributes noticeably increase the precision (2.03%) and recall (6.74%). This indicates that the two attributes are helpful in identifying the problematic features. For the performance on each app, adding the two attributes (i.e., BERT+Attr-CRF) obtains the best performance on most apps, and adding one of the two attributes (i.e., BERT+CAT-CRF or BERT+SEN-CRF) occasionally achieves the best performances on some apps (e.g., BERT+SEN-CRF on *Snapchat*). Moreover, even the performance of the base BERT-CRF model outperforms the best baseline in RQ1 (i.e., BiLSTM-CRF), which verifies the advantage of our model design.

Among the two added review attributes, the review description sentiment attribute contributes slightly more to performance improvement (1.64% in precision and 5.80% in recall) than the app category attribute (1.38% in precision and 5.26% in recall). Furthermore, we also observe that the contribution of these two attributes overlaps to some extent, i.e., the increased performance by each attribute is not simply added up to the performance of the whole model. This is reasonable considering the fact that words expressing the user sentiment could be encoded semantically in the textual descriptions and captured by the BERT model. Nevertheless, the overall performance achieved by adding both of the attributes is the highest, further indicating the necessity of our model design.

5.3 Answering RQ3

Table 5 presents the performance of SIRA in clustering problematic features, as well as the two baselines. SIRA outperforms the two

Table 3: Examples on extracted problematic features by different approaches (RQ1).

#	Review	KEFE	Caspar	SAFE	BiLSTM-CRF	SIRA
# 1	Keeps crashing when I try to take a picture of a check.	take a picture	keeps crashing, I try to take a picture of a check	keeps crashing, take a picture	take a picture of a check	take a picture of a check
# 2	When I try to view story of friend, the majority of the time it get stuck on a wheel and never load.	view story	I try to view story of friend, the majority of the time it get stuck on a wheel, never load	view story	view story	view story of friend

Table 4: Ablation experiment on attributes (RQ2).

App	Metric	BERT -CRF	BERT +CAT -CRF	BERT +SEN -CRF	BERT +Attr -CRF
Instagram	P	82.46%	**84.08%**	83.78%	83.59%
	R	80.39%	85.60%	85.50%	**85.70%**
	F1	81.34%	**84.73%**	84.56%	84.53%
Snapchat	P	**84.58%**	83.82%	83.38%	82.63%
	R	81.49%	83.31%	**85.31%**	84.15%
	F1	82.89%	83.48%	**84.23%**	83.30%
Gmail	P	88.33%	89.30%	**90.59%**	90.27%
	R	78.37%	83.43%	83.50%	**84.16%**
	F1	82.99%	86.16%	86.86%	**87.09%**
Yahoo Mail	P	75.92%	76.67%	78.23%	**79.18%**
	R	83.72%	83.72%	86.09%	**87.37%**
	F1	79.54%	79.94%	81.86%	**83.00%**
BPI Mobile	P	84.87%	85.92%	85.52%	**87.37%**
	R	78.09%	84.94%	82.60%	**85.07%**
	F1	81.25%	85.32%	83.96%	**86.13%**
Chase Mobile	P	78.24%	80.26%	80.05%	**80.32%**
	R	77.59%	82.19%	83.74%	**84.59%**
	F1	77.73%	81.11%	81.76%	**82.26%**
Overall	P	82.59%	83.73%	83.95%	**84.27%**
	R	79.69%	83.88%*	84.31%*	**85.06%***
	F1	81.10%	83.78%**	84.10%**	**84.64%***

Compared to BERT-CRF, statistical significance $p - value < 0.05$ is denoted by **, and $p - value < 0.01$ is denoted by *.

Table 5: Evaluation on problematic feature clustering (RQ3).

App	Metric	LDA	K-Means	SIRA
Instagram	ARI	0.10	**0.30**	0.29
	NMI	0.72	0.78	**0.84**
Snapchat	ARI	0.19	0.13	**0.32**
	NMI	0.80	0.72	**0.85**
Gmail	ARI	0.18	0.07	**0.45**
	NMI	0.73	0.58	**0.82**
Yahoo Mail	ARI	0.42	**0.47**	0.41
	NMI	0.81	**0.83**	0.82
BPI Mobile	ARI	0.44	0.10	**0.59**
	NMI	0.83	0.58	**0.89**
Chase Mobile	ARI	**0.38**	0.21	0.26
	NMI	0.81	0.79	**0.82**
Overall	ARI	0.21	0.14	**0.38**
	NMI	0.57	0.62	**0.77**

baselines on the overall performance, where ARI and NMI reach 0.38 and 0.77, respectively, which is higher than that of LDA (0.21 and 0.57) and K-Means (0.14 and 0.62).

Furthermore, the improvement of SIRA on ARI is greater than the improvement on NMI. ARI is a pair-wise metric, which is more sensitive when two phrases that should belong to the same cluster are wrongly assigned into different clusters, or when two phrases which should belong to different clusters are wrongly placed into the same cluster. The ARI results we obtained indicate that SIRA can effectively avoid generating new clusters or breaking up the original clusters. NMI is an entropy-based metric, which mainly focuses on the changes of two distributions based on information entropy theory. The NMI results we obtained indicate that the distribution of the entire cluster (e.g., the number of problematic features in each cluster) derived from SIRA are closer to the ground-truth.

The baseline approaches use the word statistics or co-occurrence relations to cluster the problematic features. The performance of our proposed graph-based clustering method indicates that it can better understand the semantic relations among problematic features.

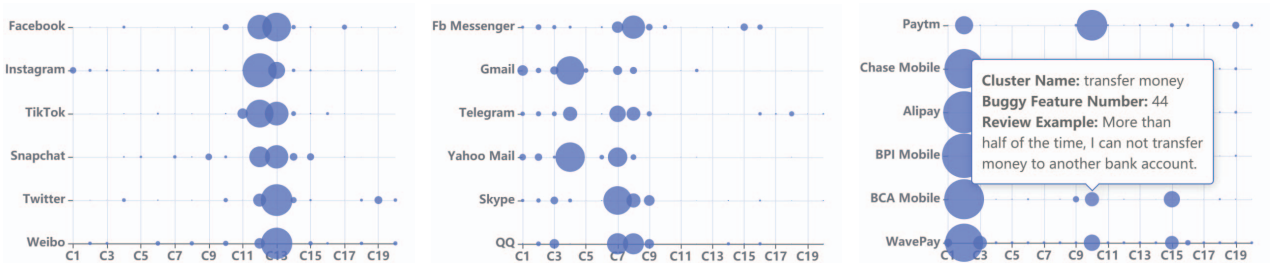
Table 6: Experimental dataset for investigating "where the apps frustrating users".

Category	App	# Reviews	# Sentences
Social	Facebook	64,559	147,156
	Instagram	63,124	153,852
	TikTok	61,178	104,094
	Snapchat	18,268	41,278
	Twitter	15,583	36,386
	Sina Weibo	10,772	37,372
Communication	Facebook -Messenger	27,121	59,303
	Gmail	9,655	24,520
	Telegram	7,704	17,672
	Yahoo Mail	7,090	20,124
	Skype	3,266	8,139
	Tencent QQ	3,194	7,326
Finance	Paytm	18,316	47,836
	Chase Mobile	3,732	9,952
	Alipay	3,153	9,359
	BPI Mobile	1,375	3,638
	BCA Mobile	386	960
	WavePay	58	124
Overall		318,534	729,091

6 WHERE THE APPS FRUSTRATE USERS - AN EMPIRICAL STUDY WITH SIRA

This section describes a large-scale empirical study with SIRA on popular apps. First, we apply SIRA to 18 apps of three categories (6 in each category) to demonstrate: 1) how SIRA can be utilized in real-world practice; 2) the distribution of problematic features across these popular apps. We also select 3 apps (1 in each category) and conduct a user survey to verify the usefulness of SIRA.

SIRA in the Large. We crawl the app reviews of 18 apps from three categories (6 in each category) submitted during February 2020 to December 2020 (note that this is different from the time period in Section 4.2). Table 6 lists the statistics of this dataset, which contains 318,534 reviews and 729,091 sentences. We run SIRA on this large-scale dataset to obtain the visualization of the clustered problematic features (see Section 3.4). In total, we obtain 113 clusters for social apps, 78 clusters for communication apps



(a) Social (b) Communication (c) Finance

Figure 4: The distribution of problematic features of different categories.

Table 7: Cluster name (i.e., representative problematic feature) of each cluster in Figure 4.

#	Social	Communication	Finance
C1	the reel option	delete email	send message
C2	like a post	open app	log in
C3	search option	receive notification	receive otp code
C4	load tweet	send and receive email	load the page
C5	use filter	dark mode	check deposit
C6	follow people	load inbox	get notification
C7	the front camera	sign into account	use finger print
C8	click on photo	send picture and video	click button
C9	send snap	video call	do transaction
C10	receive notification	see story	transfer money
C11	get live option	click on call button	get cash back
C12	post story	sync account	scan qr code
C13	access account	change the emoji and nickname	recharge mobile number
C14	open snap	share photo	change phone number
C15	send message	register user	open passbook
C16	watch video	chat with friend	book ticket
C17	dark mode	get otp for login	select option
C18	scroll the feed	receive verification code	check balance
C19	retrieve tweet	quiz bot	make payment
C20	get verification code	change phone number	receive the refund

and 90 clusters for finance apps. Figure 4 presents the visualization results of clusters for each category with the bubble size denoting the ratio of corresponding problematic features. For clarity, we only present the clusters whose number of problematic features is in top 20, by the order of cluster id. Table 7 shows the name of each cluster in Figure 4. The following observations can be obtained.

First, our visualization can provide a summarized view of the problematic features for each app and the comparison across apps. This enables the developers to acquire where the app is prone to problems, and where other apps are also likely to have issues, with a single glance. One can also derive the detailed content of each cluster, and example app reviews of the cluster by hovering the cursor over the bubble in the figure (see examples in Figure 4(c)).

Second, different apps can share similar problematic features, which can facilitate app testing and refine the testing techniques. Take Figure 4(a) as an example, although the problematic features are observed distributing differently across apps, all the six investigated apps would have a noticeable number of problematic features in certain clusters (i.e., *C12. post story* and *C13. access account*). These information can warn the developers of similar apps to notice potential problems, especially which have not yet been reported or only mentioned in a few reviews. Further, developers can leverage reviews from similar apps for quality assurance activities, rather than only focus on the limited set of reviews of its own app. This is especially the case for the less popular apps which only have few reviews regarding app problems.

Third, different apps can have their unique problematic features and problematic feature distributions, which further indicates the necessity of review mining and analysis in a fine-grained way. For example, from Figure 4(b), we can see that, based on the user reported problems, 63% reviews of the *Facebook Messenger* app relate with feature *C8. send picture and video*. By comparison, its competitor *Gmail* app is mainly prone to bugs for quite different feature *C4. send and receive email*. In addition, for its another competitor *Telegram* app, the problematic features are distributed more evenly, i.e., the number of user submitted reviews do not exert big difference across *C4*, *C7* and *C8*, and the largest cluster (i.e., *C7. sign into account*) occupies a mere of 33% reviews. From these insights provided by our approach, the developers can obtain a clear understanding of an app about the features that are prone to problems, so as to arrange the follow-up problem solving and allocate the testing activity for subsequent versions. More than that, these information can also assist the developers in the competitive analysis of apps, e.g., acquire the weakness of their app compared with similar apps.

Furthermore, a series of attempts can be made to refine the app testing techniques. For example, one can recommend problematic features to similar apps in order to prioritize the testing effort, or recommend related descriptions (mined from app reviews) to similar apps to help bug detection. In addition, the automated graphical user interface (GUI) testing techniques can be customized and the testing contents can be prioritized. Current automated GUI testing tools tend to dynamically explore different pages of a mobile app through random actions (e.g., clicking, scrolling, etc) to trigger the crash or explicit exceptions [37]. If one could know the detailed problematic features of other similar apps in advance, the explored pages can be re-ranked so that the bug-prone features can be explored earlier to facilitate the bugs being revealed earlier. We will further explore problematic features based app testing in our future work.

A User Survey. In order to assess the usefulness of SIRA, we conduct a user survey on three popular apps: *Weibo*, *QQ* and *Alipay*. We invite 15 respondents (5 from each company) in total, including 2 product managers, 5 requirement analysts, and 8 developers, who are familiar with the app reviews of their own company.

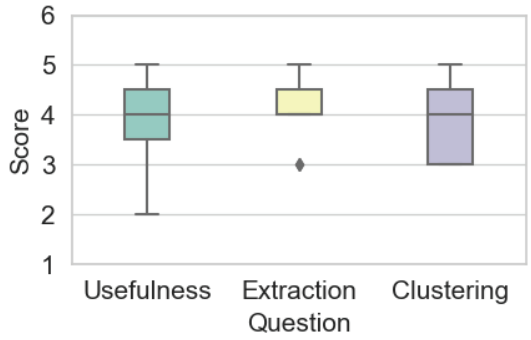


Figure 5: Feedback of user study.

More specifically, we conduct SIRA on the reviews obtained in the first week of May 2021, which contains 177 reviews from *Weibo*, 149 from *QQ*, and 177 from *Alipay* after preprocessing. Each respondent examines the extracted problematic features, clusters and visualization results obtained by SIRA, and answer the following three questions: 1) (Usefulness) Can SIRA help understand user requirements from app reviews? 2) (Extraction) Can SIRA extracted problematic features accurately? 3) (Clustering) Can SIRA cluster problematic features accurately? We provide five options for each question from 1 (strongly disagree) to 5 (strongly agree). The first question concerns the usefulness of SIRA, i.e., whether SIRA can save effort for analyzing large-scale app reviews. The last two questions concern the performance of SIRA on problematic feature extraction and clustering respectively, when analyzing app reviews in real-world practice.

Figure 5 shows the box plot statistics of respondents' feedback. There are respectively 11, 13 and 10 (out of 15) respondents give the score over 3 for Q1, Q2, and Q3. Most of them (over 73%) are satisfied (score over 3) with the usefulness of SIRA, and think SIRA can help them obtain a fine-grained understanding on problematic features. The average score of Q1, Q2, and Q3 are 3.93, 4.13, and 3.93 respectively. Besides, three of them heard about or tried existing review analysis tools such as INFAR [12] and SUR-Miner [15], and they admit the advantages of SIRA as its extracted features and derived clusters are finer-grained and more meaningful. We also interviewed the respondents about the possible enhancement of SIRA. They said there were still some cases where SIRA doesn't work well, such as some extracted phrases contain two or more features, which leads to poor performance of clustering. This can be solved in future work by exploring the patterns of such tangled features and deconstructing them into separate ones. In addition, we received some suggestions from developers for better visualizations (e.g., supporting interactive visual analytics).

7 DISCUSSION

Advantage Over Topic Discovery Approaches. There are several previous approaches which involve topic discovery [12, 15, 52, 57, 58]. Yet, their discovered topics are more coarse-grained than our proposed approach. For example, based on 95 mobile apps like *Facebook* and *Twitter* from Google Play, MARK [57] can only discover such topics as *crash*, *compatibility*, and *connection*, and PUMA [58] generates topics like *battery consumption*. Similarly, SUR-Miner [15] generates topics such as *predictions*, *auto-correct*, and *words*. SURF [52] can discover topics such as *GUI*, *app*, and *company*, while INFAR [12] can generate topics like *update*, *radar*, *download*. With these discovered topics, the developers can acquire a general view about the problems the app undergoes, yet could not get a clear understanding about where it is wrong. By comparison, as demonstrated in Figure 4 and Table 7, our proposed approach can generate more finer-grained topics as *open message, get crash back*, which helps developers achieve a deeper and more accurate understanding about where the app is wrong.

Threats to Validity. The ***external threats*** concern the generality of the proposed approach. We train and evaluate SIRA on the dataset consisting of six apps from three categories. The selected apps and their belonging categories are all the commonly-used ones with rich reviews in practice, which relatively reduces this threat. In addition, we demonstrate the usage of SIRA on a much bigger dataset derived from 18 apps. The results are promising, which verifies its generality further. Regarding ***internal threats***, SIRA is a pipeline method, where the problematic feature clustering depends on the accuracy of extracting problematic features. Since we have seen a relatively high performance of SIRA on problematic feature extraction, we believe SIRA can alleviate the error accumulation to some extent. In addition, we reuse the source code from the original paper (i.e., for *Caspar* and *KEFE*), or the open source implementation (i.e., for *SAFE*, *K-Means*, and *LDA*) for the baselines, which help ensure the accuracy of the experiments. The ***construct validity*** of this study mainly questions the evaluation metrics. We utilize precision, recall and F1-Score to evaluate the performance of problematic feature extraction. We consider that a problematic feature is correctly extracted when it is the same as the ground-truth, which is a rather strict measure. The metrics used to evaluate clustering results are also commonly used [24].

8 CONCLUSION

To help acquire a concrete understanding about where the app is frustrating the users, this paper proposes a semantic-aware, fine-grained app review analysis approach SIRA, which can extract, cluster, and visualize the problematic features of app reviews. SIRA designs a novel BERT+Attr-CRF model to extract fine-grained problematic features, and employs a graph-based clustering method to cluster them. We evaluate SIRA on 3,426 reviews from six apps, and the results confirm the effectiveness of the proposed approach. We further conduct an empirical study on 318,534 reviews from 18 popular apps to explore its potential application and usefulness in real-world practice. Our source code and experimental data are publicly available at: https://github.com/MeloFancy/SIRA.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under grant No.2018YFB1403400, the National Natural Science Foundation of China under grant No.62072442, the Youth Innovation Promotion Association Chinese Academy of Sciences, and Australian Research Council Discovery Project DP220103044.

REFERENCES

[1] 2014. Viterbi Algorithm. https://en.wikipedia.org/wiki/Viterbi_algorithm.

[2] Raul Berrios, Peter Totterdell, and Stephen Kellett. 2015. Eliciting mixed emotions: a meta-analysis comparing models, types, and measures. *Frontiers in psychology* 6 (2015), 428.

[3] C. Biemann. 2006. Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. *Association for Computational Linguistics* (2006).

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2001. Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani (Eds.). MIT Press, 601–608.

[5] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, Eduardo Blanco and Wei Lu (Eds.). Association for Computational Linguistics, 169–174.

[6] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 767–778.

[7] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.* 12 (2011), 2493–2537.

[8] Nathan Cooper, Carlos Bernal-Cárdenas, Oscar Chaparro, Kevin Moran, and Denys Poshyvanyk. 2021. It Takes Two to Tango: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports. In *International Conference on Software Engineering (ICSE)*.

[9] Hong-Jie Dai, Po-Ting Lai, Yung-Chun Chang, and Richard Tzong-Han Tsai. 2015. Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization. *J. Cheminformatics* 7, S-1 (2015), S14.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186.

[11] J Felsenstein and G A Churchill. 1996. A Hidden Markov Model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution* 13, 1 (01 1996), 93–104.

[12] Cuiyun Gao, Jichuan Zeng, David Lo, Chin-Yew Lin, Michael R. Lyu, and Irwin King. 2018. INFAR: insight extraction from app reviews. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. 904–907.

[13] Cuiyun Gao, Jichuan Zeng, Michael R. Lyu, and Irwin King. 2018. Online app review analysis for identifying emerging issues. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 48–58.

[14] Cuiyun Gao, Jichuan Zeng, Xin Xia, David Lo, Michael R. Lyu, and Irwin King. 2019. Automating App Review Response Generation. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 163–175.

[15] Xiaodong Gu and Sunghun Kim. 2015. "What Parts of Your Apps are Loved by Users?". In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*. 760–770.

[16] Hui Guo and Munindar P. Singh. 2020. Caspar: extracting and synthesizing user stories of problems from app reviews. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*. 628–640.

[17] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically enhanced software traceability using deep learning techniques. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*. 3–14.

[18] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. 2017. An exploratory study of Twitter messages about software applications. *Requir. Eng.* 22, 3 (2017), 387–412.

[19] Emitza Guzman and Walid Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25-29, 2014*, Tony Gorschek and Robyn R. Lutz (Eds.). IEEE Computer Society, 153–162.

[20] Rui Hao, Yang Feng, James Jones, Yuying Li, and Zhenyu Chen. 2019. CTRAS: Crowdsourced test report aggregation and summarization. In *ICSE'2019*. 921–932.

[21] Mark Harman, Yue Jia, and Yuanyuan Zhang. 2012. App store mining and analysis: MSR for app stores. In *9th IEEE Working Conference of Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland*. 108–111.

[22] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, 328–339.

[23] Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. 2014. Effective Bug Triage Based on Historical Bug-Fix Information. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. 122–132.

[24] Yi Huang, Chunyang Chen, Zhenchang Xing, Tian Lin, and Yang Liu. 2018. Tell them apart: distilling technology differences from crowd-scale comparison discussions. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 214–224.

[25] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR* abs/1508.01991 (2015).

[26] Md Rakibul Islam and Minhaz F. Zibran. 2018. SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text. *J. Syst. Softw.* 145 (2018), 125–146.

[27] Timo Johann, Christoph Stanik, Alireza M. Alizadeh B., and Walid Maalej. 2017. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*. 21–30.

[28] Wang Junjie, Cui Qiang, Wang Song, and Wang Qing. 2017. Domain Adaptation for Test Report Classification in Crowdsourced Testing. In *ICSE'17*. 83–92.

[29] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[30] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. 2015. What Do Mobile App Users Complain About? *IEEE Softw.* 32, 3 (2015), 70–77.

[31] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[32] Ron Kohavi. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. 1137–1145.

[33] Zijad Kurtanovic and Walid Maalej. 2018. On user rationale in software engineering. *Requir. Eng.* 23, 3 (2018), 357–379.

[34] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, Carla E. Brodley and Andrea Pohoreckyj Danyluk (Eds.). Morgan Kaufmann, 282–289.

[35] Lawrence, HubertPhipps, and Arabie. 1985. Comparing partitions. *Journal of Classification* (1985).

[36] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. 926–931.

[37] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: a lightweight UI-guided test input generator for Android. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*. 23–26.

[38] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. 2019. Pattern-based mining of opinions in Q&A websites. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 548–559.

[39] Hui Liu, Mingzhu Shen, Jiahao Jin, and Yanjie Jiang. 2020. Automated classification of actions in bug reports of mobile apps. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*. 128–140.

[40] Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015*. 116–125.

[41] Yichuan Man, Cuiyun Gao, Michael R. Lyu, and Jiuchun Jiang. 2016. Experience Report: Understanding Cross-Platform App Issues from User Reviews. In *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*. 138–149.

[42] William J. Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2017. A Survey of App Store Analysis for Software Engineering. *IEEE Trans.*

Software Eng. 43, 9 (2017), 817–847.

[43] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. 2000. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, Pat Langley (Ed.). Morgan Kaufmann, 591–598.

[44] Xuan Vinh Nguyen, Julien Epps, and James Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *J. Mach. Learn. Res.* 11 (2010), 2837–2854.

[45] Ehsan Noei, Daniel Alencar da Costa, and Ying Zou. 2018. Winning the app production rally. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. 283–294.

[46] Emanuel Oehri and Emitza Guzman. 2020. Same Same but Different: Finding Similar User Feedback Across Multiple Platforms and Languages. In *28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020*, Travis D. Breaux, Andrea Zisman, Samuel Fricker, and Martin Glinz (Eds.). IEEE, 44–54.

[47] Fabio Palomba, Mario Linares Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2015. User reviews matter! Tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*. 291–300.

[48] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*. 281–290.

[49] Ian Porteous, David Newman, Alexander T. Ihler, Arthur U. Asuncion, Padhraic Smyth, and Max Welling. 2008. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, Ying Li, Bing Liu, and Sunita Sarawagi (Eds.). ACM, 569–577.

[50] Lev-Arie Ratinov and Dan Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning, CoNLL 2009, Boulder, Colorado, USA, June 4-5, 2009*, Suzanne Stevenson and Xavier Carreras (Eds.). ACL, 147–155.

[51] Gerard Salton and Michael McGill. 1984. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company.

[52] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*. 499–510.

[53] Aonan Tang, David Jackson, Jon Hobbs, Wei Chen, Jodi L. Smith, Hema Patel, Anita Prieto, Dumitru Petrusca, Matthew I. Grivich, Alexander Sher, Pawel Hottowy, Wladyslaw Dabrowski, Alan M. Litke, and John M. Beggs. 2008. A Maximum Entropy Model Applied to Spatial and Temporal Correlations from Cortical Networks In Vitro. *Journal of Neuroscience* 28, 2 (2008), 505–518.

[54] Gias Uddin and Foutse Khomh. 2017. Opiner: an opinion search and summarization engine for APIs. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen (Eds.). IEEE Computer Society, 978–983.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008.

[56] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*. 14–24.

[57] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. 2015. Mining User Opinions in Mobile App Reviews: A Keyword-Based Approach. In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*. 749–759.

[58] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. 2016. Phrase-based extraction of user opinions in mobile app reviews. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, David Lo, Sven Apel, and Sarfraz Khurshid (Eds.). ACM, 726–731.

[59] Han Wang, Chunyang Chen, Zhenchang Xing, and John C. Grundy. 2020. DiffTech: a tool for differencing similar technologies from question-and-answer discussions. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. 1576–1580.

[60] Junjie Wang, Mingyang Li, Song Wang, Tim Menzies, and Qing Wang. 2019. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. *Inf. Softw. Technol.* 110 (2019), 139–155.

[61] Huayao Wu, Wenjun Deng, Xintao Niu, and Changhai Nie. 2021. Identifying Key Features from App User Reviews. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 922–932.

[62] Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving Automated Bug Triaging with Specialized Topic Model. *IEEE Trans. Software Eng.* 43, 3 (2017), 272–297.

[63] Liang Xu, Qianqian Dong, Cong Yu, Yin Tian, Weitang Liu, Lu Li, and Xuanwei Zhang. 2020. CLUENER2020: Fine-grained Name Entity Recognition for Chinese. *arXiv preprint arXiv:2001.04351* (2020).

[64] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*. 1385–1397.

[65] Wei Zhang, Hong Mei, and Haiyan Zhao. 2006. Feature-driven requirement dependency analysis and high-level software design. *Requir. Eng.* 11, 3 (2006), 205–220.

[66] Yuan Zhang, Hongshen Chen, Yihong Zhao, Qun Liu, and Dawei Yin. 2018. Learning Tag Dependencies for Sequence Tagging. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, Jérôme Lang (Ed.). ijcai.org, 4581–4587.

[67] Jian Zhou and Hongyu Zhang. 2012. Learning to Rank Duplicate Bug Reports. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. Association for Computing Machinery, New York, NY, USA, 852–861.