2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

Emerging App Issue Identification from User Feedback: Experience on WeChat

Cuiyun Gao[†], Wujie Zheng[§*], Yuetang Deng[§], David Lo[‡], Jichuan Zeng[†], Michael R. Lyu[†], Irwin King[†]
[†]Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, China
[‡]School of Information Systems, Singapore Management University, Singapore
[§]Tencent, Inc., China
{wujiezheng,yuetangdeng}@tencent.com, {cygao,jczeng,lyu,king}@cse.cuhk.edu.hk, davidlo@smu.edu.sg

***Abstract*—It is vital for popular mobile apps with large numbers of users to release updates with rich features while keeping stable user experience. Timely and accurately locating emerging app issues can greatly help developers to maintain and update apps. User feedback (*i.e.*, user reviews) is a crucial channel between app developers and users, delivering a stream of information about bugs and features that concern users. Methods to identify emerging issues based on user feedback have been proposed in the literature, however, their applicability in industry has not been explored. We apply the recent method IDEA to WeChat, a popular messenger app with over 1 billion monthly active users, and find that the emerging issues detected by IDEA are not stable (*i.e.*, due to its inherent randomness, its results change when run multiple times even for the same inputs), and there are other problems such as long running time. To address these limitations, we design a novel tool, named DIVER. Different from IDEA, DIVER is more efficient (it can report real-time alerts in seconds), generates reliable results, and most importantly, achieves higher accuracy in our practice. After its deployment on WeChat, DIVER successfully detected 18 emerging issues of WeChat's Android and iOS apps in one month. Additionally, DIVER significantly outperforms IDEA by 29.4% in precision and 32.5% in recall.**

***Index Terms*—Mobile apps, app reviews, emerging issue detection, anomaly**

I. Introduction

In 2018, a bulk of apps have been published on app markets (*e.g.*, 3.8 million apps on Google Play and 2.0 million on App Store [1]). For an app to become competitive and prevalent, user-friendly design and rapid responsiveness are crucial factors. Timely identification of bugs and unsatisfactory features that affect many users is important for app developers during app testing and maintenance process. For example, in July of 2016, Pokémon Go, a popular game app, was flooded with one-star ratings on app markets (*e.g.*, 25,000 out of 55,000 in App Store), when an updated version was released with tracking features removed [2]. Such situation could be alleviated if thorough testing was implemented before final release, or immediate fix was conducted after release. For WeChat[1], a highly popular messenger app (especially among users of Chinese origins) released by Tencent, Inc., its large audience (around 1.04 billion monthly active users [3]), complex functionalities[2], and frequent updates drive the need for timely identification of emerging issues to ensure its reliability and user satisfaction.

However, timely and accurate detection of critical app issues is challenging in industrial scenario. Before app release, the typical way to achieve a high-quality app is by performing thorough testing [4], but the testing process tends to be labor-intensive and time-consuming in practice [5]–[7]. According to [8], popular apps usually update their versions on a bi-weekly basis or more frequently. Also, due to the complicated app functionalities and fragmentation issues, not all the features or usage paths could be covered during testing [7].

Another way to detect issues is based on user feedback analysis. User feedback is directly written by users based on their experience of an app, and reflects the app's major bugs and annoying features. Such resource can be easily collected either from testing users during the beta testing phase or all the users after release. There already exist some prior works on mining user feedback for assisting app evolution and maintenance [9], [10]. According to previous studies [11], [12], developers who consider user reviews are rewarded in terms of higher user ratings of their apps. Unfortunately, the large quantity of user reviews (*e.g.*, WeChat receives around 60,000 reviews per day) makes manual analysis inefficient and unrealistic. Moreover, reviews usually contain much noisy information such as non-informative reviews [13]. These characteristics of user feedback increase the difficulties of automatic detection of emerging issues.

To our best knowledge, IDEA [14] is the most recent work that can be directly applied to detect emerging issues from user feedback. The input of IDEA is user reviews distributed in consecutive app versions. Based on topic modeling, which is a typical approach to infer topic structure of a collection of documents, IDEA outputs emerging app issues in the level of phrases and sentences. However, we met several problems when applying it in practice in WeChat. The first problem is the *stability* of the running results. Random initialization of parameters in topic modeling makes the detected emerging

* Wujie Zheng is the corresponding author.

[1]https://www.wechat.com/en/

[2]WeChat has now evolved to be well beyond a messenger app: it also provides functionalities such as banking, shopping, playing games, news browsing, and serves as a platform for third parties to develop their own apps.

978-1-7281-1760-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-SEIP.2019.00040

issues not exactly the same after each run. Second, manually choosing appropriate *topic numbers* for different datasets is difficult, where the choice of topic number can easily impact the accuracy of detected emerging issues [15], [16]. The third problem is related to its *efficiency*. According to [14], IDEA can process 160 reviews per second, meaning that analyzing 60,000 reviews generated by WeChat in one day may need at least 6 minutes for each trial. For achieving good performance, we usually need to tune hyper-parameters of topic modeling, which requires processing the data many times [16]. Such time cost is not ideal enough for popular apps, where more time spent on detecting important issues usually means more customer churn.

In this paper, we try to solve the above problems existing in IDEA and build a new tool named DIVER, *i.e.*, i**D**entifying emerging app **I**ssues **V**ia us**ER** feedback. The input of DIVER is similar to that of IDEA, that is, user reviews posted in different time slices or for different versions. One example of DIVER's output is shown in Fig. 1, with app issues explained in related keywords and feedback sentences. To facilitate developers' issue checking and fixing, the emerging issues are sent to developers by email and WeChat immediately after being captured. After the deployment of DIVER on WeChat, DIVER helped developers identify 18 emerging issues of its Android app and iOS app in the January of 2018. To further verify the effectiveness of DIVER, we conduct evaluation experiments on the WeChat apps for Android and iOS. Experimental results demonstrate that DIVER significantly outperforms IDEA by 29.4% in precision and 32.5% in recall on average. Moreover, DIVER can process thousands of user reviews in seconds, which is more applicable for the agile development mode in industry.

The paper makes the following contributions: (1) We build a tool called DIVER to identify emerging app issues from user feedback more effectively and efficiently. The tool can help the beta testing process before official release or the maintenance process after release; (2) We conduct comprehensive experiments on industry apps to verify the effectiveness of DIVER; and (3) We have implemented and deployed DIVER to monitor reviews of popular apps in the Tencent company, especially WeChat apps.

The rest of this paper is organized as follows. In Section II, we introduce the motivation and background of this work. We present related work in Section III. The detailed methodology of DIVER is described in Section IV, with experiment setup elaborated in Section V and experimental results presented in Section VI. In Section VII, we provide a qualitative analysis of successful and unsuccessful cases when DIVER was applied to industry apps. We discuss limitations and conclude this paper in Section VIII and Section IX, respectively.

II. Motivation and Background

In practical app development, managers must manage well the time interval between issue detection and version update to ensure enough time for app modification. According to our industrial partners' experience, although the automated testing process in WeChat can cover around 80% of critical app features, these app testing methods can be limited due to the complicated functionalities in WeChat, wide range of devices to test, and many versions to be tested at the same time. Developers can well monitor crashes and performance issues by analyzing logs related to crashes and Application Not Responding (ANR) issues. But the functionality- and interface-related issues are not easy to be identified. Different from app logs, user feedback contains various app facets that concern the users. Thus, analyzing user feedback is an important activity to ensure that developers are not ignorant of user needs.

2018-02-10 10:03, Version ID 232, User No.: 3234
Warning 1
Keywords: Sound, Video
Feedback No.: 5
Sample Related Feedback:
① The ***video sound*** cannot be closed, crash! (from user_1)
② Why isn't there any ***sound*** in the ***video*** sent by my friends? I've already set it in my phone. (from user_2)
③ There is no ***sound*** in my ***video*** and voice messages. (from user_3)
④ The ***video*** sent from my friends doesn't have any ***sound***. (from user_4)

Fig. 1: Example warning captured by DIVER.

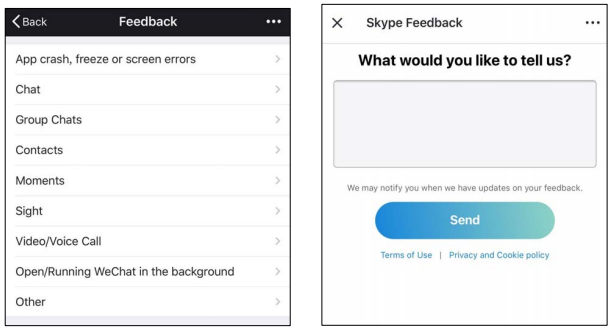
To assist readers in better understanding the industrial scenario of DIVER, in this section, we first briefly introduce what user feedback is, and then explain the importance of identifying emerging app issues.

A. User Feedback

Developers can collect user feedback through apps or app markets (*e.g.*, Google Play and App Store). Fig. 2 illustrates the feedback submission forms of WeChat and Skype respectively. For WeChat, users can first choose a relevant topic and then write detailed problems encountered or submit screen shots (*e.g.*, the feedback tree shown on the left in Fig. 2). WeChat also provides a channel for users to write their comments without choosing a topic, similar to Skype feedback submission style (shown on the right in Fig. 2). Besides review texts, user reviews are generally accompanied by other attributes, such as user name or identifier, post time, app version, device type, and system type (*e.g.*, iOS or Android). Review texts can describe everything that users care about, including user requirements, potential app bugs, and features to improve. For example, one user of the Instagram app, which is a social networking app, complains about a crash issue by writing that "*App is absolute trash. Crashes every three seconds on a flagship stock Pixel 2 XL with zero problems with any other quality app.*"

B. Emerging App Issues

For an app, the numbers or proportions of app issues reflected in user feedback are normally stable over time. But when a new version is released, the number or proportion of feedback associated with certain functionalities or bugs may sharply increase. Such functionality- or bug-related issues are emerging app issues. An example of emerging issue is depicted in Fig. 3. After releasing WeChat version X on July



(a) Feedback tree (b) Feedback form

Fig. 2: Feedback submission forms of WeChat (left) and Skype (right).

5, 2017, the volume of user comments[3] related to "*sound*" rises abruptly. This is because version X had a functionality error in sharing sound and pictures which was not fixed when it was released. Before the issue was detected on July 7, the issue has been raised as feedback by a large number of users (from around 1,000 to 2,000+). We can see that the longer the delay in finding emerging issues, the more risk an app faces, as some users may uninstall the app during the period. Finally, the issue was solved in a revised version on July 7, and the number of feedback related to "*sound*" started to decline.

As WeChat Android and iOS apps receive a large number of feedback by many users during beta testing and release phases, some functionality issues may take a long time to be spotted by developers. Thus, additional tool support is needed to accurately identify emerging app issues in a timely fashion which is critical to ensure user satisfaction.

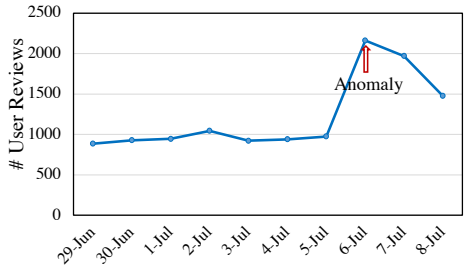


Fig. 3: Number of user reviews related to "*sound*" for all WeChat versions from June 29 - July 8, 2017.

III. Related Work

Two threads of work on user feedback analysis inspire the design of our proposed approach DIVER, including user intention mining and emerging issue detection.

A. User Intention Mining

App reviews serve as a communication bridge between developers and users. However, review mining task is inherently challenging due to the noisy nature of user review data.

[3]Note that the user comments considered are from all the current versions.

User intention mining aims at accurately understanding the topics delivered by reviews, *e.g.*, whether the user is complaining about the interface or performance cost of an app. There already exist many attempts on analyzing user intentions from user reviews, as summarized by Martin *et al.* [17]. Iacob *et al.* [18] manually analyze 3,278 reviews of the apps in Google Play and summarize nine recurring themes among feedbacks. They find that major bugs usually trigger additional negative feedback, which can support app testing. Khalid *et al.* [19] manually tag 6,390 low star-rating reviews from iOS apps and reach the conclusion that the most frequent complaints are related to functionality errors, feature requests, and app crashes. Driven by the increasing amount and importance of user reviews, there exists research effort [9], [20]–[24] aiming at automating the review tagging process. For example, Iacob and Harrison [25] automatically extract the reviews related to feature requests based on predefined linguistic rules. Chen *et al.* [13] focus on prioritizing the informative reviews by employing topic modeling methods. Maalej and Nabil [22] supervisedly classify reviews into four types such as bug reports and feature requests. As labeling reviews usually cost lots of manual labor, Di Sorbo *et al.* [24] propose an unsupervised method to classify reviews to predefined topics. To deeply understand users' attitude towards specific app issues, Guzman *et al.* [26] automatically extract keyphrases from user reviews, and predict their sentiment based on existing sentiment analysis techniques. Similarly, Gu and Kim [27] improve the performance of aspect opinion extraction. In spite of the effectiveness of these prior studies on user review understanding, reviews' version-sensitive characteristics are not well or deeply explored [14], [28], [29].

B. Emerging Issue Detection

Identifying emerging app issues from user reviews is a challenging task due to the lack of real-word datasets with labels. Most current work focuses on observing the trends of app issues over time [30], and determining the emerging issues based on traditional anomaly detection methods [31]. For example, Vu *et al.* [32] detect sudden issues by counting the most related keywords. Since single word may be ambiguous without contexts, their follow-up work [33] proposes a phrase-based clustering approach, where the phrase template mining process is time-consuming and labor-intensive due to the manual validation of part-of-speech (PoS) sequences. To explore the effects of user reviews on version changes, Gao *et al.* [14] introduce IDEA to detect emerging issues of current versions based on statistics of previous versions. They present a topic labeling approach to automatically interpret topics with phrases and sentences. Although the method is free of labor cost, the performance of their work could be easily influenced by the inborn limitations of topic modeling, such as the predefined topic numbers, random initialization of topic centers, and review quantity [15]. We can also follow the methods in [22]–[24], [34] to classify reviews first and then observe the trends of these topics. However, these methods need either labeled reviews for training or predefined topics. In

practice, developers usually require more detailed and flexible topics to guide their issue checking and fixing.

IV. Methodology and Implementation of DIVER

In this section, we elaborate the workflow of DIVER. Fig. 4 presents a high-level architecture of DIVER, which includes five major steps: (1) preprocessing user reviews, (2) extracting word collocations, (3) identifying emerging word collocations, (4) grouping emerging collocations with similar topics and ranking representative feedback for each topic, and (5) issue visualization. The details of each step are provided as follows.

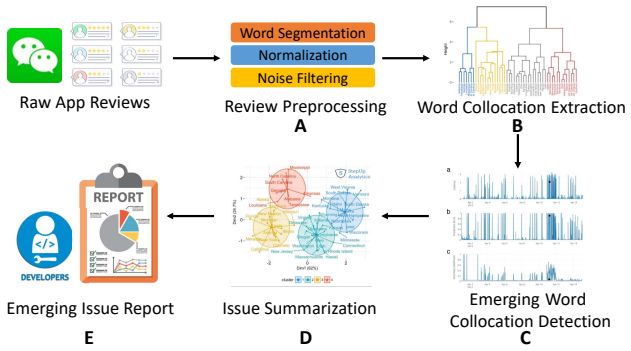


Fig. 4: The overview of DIVER.

A. Review Preprocessing

The collected reviews in WeChat are written mainly in Chinese and English. In the light of structure difference between the two languages (*e.g.*, Chinese sentences do not contain spaces to separate single words), we conduct slightly different preprocessing procedures according to language types. For Chinese feedback, we first employ word segmentation tool *jieba* [35] to produce segmented sentences. Then we format/normalize texts for both languages by removing punctuation and stop words provided by NLTK[4], filtering noisy words, such as predefined non-informative words (*e.g.*, "*hello*" in English or "你好" in Chinese), and emotional words (*e.g.*, "amazing" in English or "不错" in Chinese). We use the list of non-informative English words defined in [14] and manually create a similar list of meaningless Chinese words. To further guarantee the quality of word collocations extracted in the subsequent procedure, we also remove consecutively duplicate words, *e.g.*, "Very very new" is converted into "Very new".

B. Word Collocation Extraction

The meaning of a single word may be ambiguous. For example, the word "*sound*" in a comment can indicate "*sound notification*" or "*audio sound*" for social media apps such as WhatsApp, Line and WeChat. In contrast, issues presented via phrases are semantically more accurate. However, user reviews are usually short and unstructured, thus extracting consecutive words as phrases is sensitive to the caprices of various writing styles [36]. For instance, consider two app reviews, one containing "*The audio has no sound*" and the other containing "*Can't hear the audio sound*", it is clear that they are discussing the same topic "*audio sound*". Although "*audio*" and "*sound*" are not two consecutive words in the first review example, "*audio sound*" is a meaningful phrase. Within the context of this task, we define the *phrase* as an unordered set of words appearing in the same review.

As shown in Fig. 5, we employ frequent pattern mining approach here to extract the frequently co-occurring words, *i.e.*, phrases, together with all the frequent words as candidate terms for the next step.

Traditional frequent pattern mining algorithms, such as Apriori [37], are computational expensive and hard to scale. For WeChat, there are over 60,000 reviews received every day and over 500,000 unique words in those reviews. It is difficult or even impossible to directly apply Apriori to retrieve all the co-occurring words in the industry scenario because of the huge search space involved in finding them. Therefore, we employ ECLAT (Equivalence Class Transformation) [38], a depth-first search algorithm for pattern mining that is considered much faster and memory efficient than Apriori algorithm. With the help of ECLAT, we can obtain all the candidate phrases whose frequencies are larger than a user-defined support threshold.

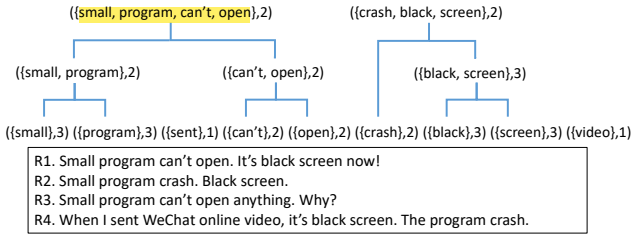


Fig. 5: Example of word collocation extraction. The hierarchical word collocation structure is built from the reviews (R1-R4), based on the co-occurrences of words in the reviews. The number after each word collocation (inside the brace) indicates the occurrence frequency of the collocation.

C. Emerging Word Collocation Detection

In order to detect the emerging issues, we need to figure out the word collocations for which the volume of reviews containing them increases significantly.

At an immediate time period after version release, some common words describing app updating may increase dramatically in quantity or proportion (e.g., "*update*" and "*download*"), but they are not emerging issues and may lead to false positive results. Here, we propose a multidimensional emerging word collocation detection, which can detect word collocation changes at a fine granularity and avoid the above-mentioned false positives. We organize our review data considering three dimensions: word collocation, version, and time. As shown in Fig. 6, for each word collocation, we extract the proportion of feedback containing it for each version, at each time unit (*e.g.*, day) after the version release date. We

[4]https://www.nltk.org/

then determine emerging word collocations by comparing the changes in the proportions comparing prior versions (for the same time unit) and prior time units (for the same version). To mitigate the bias of considering just one previous version or time unit, we introduce a moving window to involve more historical versions (indicated by the dark green block in Fig. 6), or more historical time units (indicated by the light orange block in Fig. 6). In the following paragraphs, we introduce the comparison methods from version and time dimensions respectively.

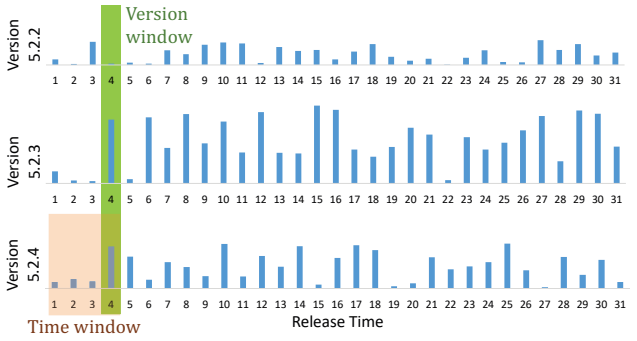


Fig. 6: Illustration of determining whether one collocation is emerging or not. The horizontal axis is the version release time (time unit = day here). The vertical axis of each bar chart is the proportion of the collocation. The light orange rectangle is the moving time window w involving records of the first four release days for Version 5.2.4, *i.e.*, $w = 4$. The dark green rectangle shows the moving version window τ including statistics of the fourth release day for the three example versions, *i.e.*, $\tau = 3$. We can see that the collocation is deemed as an emerging one on the fourth release day for Version 5.2.3, but not for Version 5.2.4, since the proportion for Version 5.2.3 is significantly larger than that of its previous version 5.2.2, and its previous release days.

1) Version-Based Comparison: We denote the percentage of the collocation at the t-th release time unit and for version v as $P_v(t)$, and the average percentage in all the considered historical versions as $\overline{P_\tau(t)}$, where τ is the version window size. A collocation is considered to be potentially emerging if it satisfies the following inequality:

$$\frac{P_v(t) - \overline{P_\tau(t)}}{\sigma(t)} > \gamma, \tag{1}$$

where $\sigma(t)$ is the standard deviation of the percentages in the version window, and γ indicates a user-defined threshold. The threshold γ determines how far the collocation's percentage for the current version differs from the expected percentage as compared to the typical difference (*i.e.*, the standard deviation). In statistics, a relative deviation of 1.2 (*i.e.*, an actual difference of 1.2 standard deviations) is significant, which has 11.6% acceptance rate[5] [39] and works well in our application. Therefore, if $\gamma > 1.2$, DIVER will regard the collocation as one that potentially has a sudden and significant change in the trend for the corresponding version. The value of this threshold (which is set by default to 1.2) can be adjusted by users.

[5]Acceptance rate 11.6% means that we accept 11.6% of the total word collocations as emerging ones.

2) Time-Based Comparison: For time-based comparison, we compute the average percentage in all the considered historical time units and denote it as $\overline{P_w(t)}$, where w is the time window size. We use a similar formula for detecting emerging collocations as shown in Eq. 1, but replace $\overline{P_\tau(t)}$ with $\overline{P_w(t)}$.

Word collocations that are identified as emerging based on both version- and time-based comparisons are outputted to the next step.

D. Issue Summarization

The emerging issues detected from the last procedure may contain semantically similar collocations. The redundancy of word collocations often results in semantically-incomplete anomaly warnings. For instance, both words "*data*" and "*recovery*" exhibit emerging trends in one version. Without grouping the words, developers would feel confused about the issue meaning when observing each emerging issue separately. To condense the emerging issue descriptions, we first cluster the words delivering the same topic and then rank the reviews in each topic for facilitating developers' observation.

1) Emerging Collocation Clustering: To cluster emerging collocations, we need to first compute semantic representation of each word collocation and then measure the relevance among them. Basically, words' semantics are embodied by their contexts. Thus, the word representations can be learned by the subordinate reviews intuitively. Our approach is based on vector space model [40], widely-used to represent textual documents in information retrieval systems.

Based on vector space model, we represent each word collocation by a vector, where the vector length is determined by the review number in the collection. For each review, corresponding to each element in the vector, we calculate the *tf.idf* (term frequency - inverse document frequency [40]) value. The term frequency (*tf*) of one collocation in a review is the number of its occurrences in that review, while the document frequency (*df*) of a collocation is the number of reviews containing it in the collection. We compute the *tf.idf* weight for a collocation as:

$$tf.idf = \frac{tf}{\log(df + 1)}. \tag{2}$$

Based on the calculated *tf.idf* weights for all reviews, we obtain a representative vector for each collocation. Similarity measurement between collocation is calculated by their cosine distance.

To automatically determine the cluster number, we exploit agglomerative hierarchical clustering approach. We denote each detected emerging collocation as p, and distance matrix $D = \mathbf{1} - M$, where M is the cosine similarity matrix of collocations. The clustering process is depicted in Algorithm 1. We start by initializing an active set A, in which each cluster

comprises one collocation. Then we merge two clusters that present minimum distance m, where the distance between two clusters is calculated as the minimum distance between the collocation pairs in the clusters. The clustering process will stop when minimum distance m achieves a defined threshold ε (which is empirically set to 0.1 in our implementation) or all the emerging issues are fused into one cluster.

Algorithm 1: Anomaly Clustering Algorithm

```
function Clustering ({p_i}_{i=1}^N, D, ε);
A ← ∅, m ← 1
for i ← 1 ··· N do
    A ← A ∪ {{p_i}}
end
while |A| > 1 and m > ε do
    c*_1, c*_2 ← argmin_{c1,c2∈A, c1≠c2} M_{c1,c2}
    m ← M_{c*_1,c*_2}
    c* ← c*_1 ∪ c*_2
    for c ∈ A do
        M_{c*,c} = M_{c,c*} = min(M_{c*_1,c}, M_{c*_2,c})
    end
    Del M_{c*_1,·}, M_{c*_2,·}, M_{·,c*_1}, M_{·,c*_2}
    A ← (A \ c*_1) \ c*_2
    A ← A ∪ c*
end
return Cluster A
```

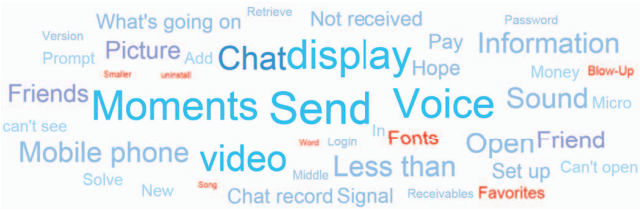
2) Review Ranking: Due to limited contextual information, words or phrases are usually not able to completely deliver users' intents. To assist developers in understanding the emerging issues efficiently, we also recommend representative user reviews for each emerging issue cluster. We treat the collocations in one cluster as query q, and employ vector space model to retrieve most relevant user reviews. For each review d, we compute its proximity $Score(d)$ to query q by

$$Score(d) = \sum_{w \in q \cap d} tf.idf_{w,d}, \tag{3}$$

where $tf.idf_{w,d}$ is obtained by Equation (2). Reviews with larger scores are more relevant to the anomaly cluster, and will be ranked higher. We choose top five user reviews for developers' reference.

E. Emerging Issue Report

The last step of DIVER is to visualize the detected emerging issues and prioritized user feedback. For explicitly illustrating the topics that users are talking about in reviews, we employ word cloud to present commonly-used words, where word sizes are determined by their proportions in the collected data. Fig. 7 depicts an example of commonly-used words written by users, with the emerging words highlighted in red. When developers move their mouse over specific words, they can observe the corresponding proportions and growth rate comparing the current time slot with the previous time slot,



Feedback Keyword	User Volume	Percentage(%)	Percentage Fluctuation
Display	502	7.59	+0.09%
Voice	380	5.74	-0.71%
Fonts	34	0.51	+1.21%
.....	.....	.....	.....

Fig. 7: Visualization of DIVER's results. Larger-sized fonts in the word cloud indicate that the corresponding words appear more often in the collection, and red fonts denote the detected emerging issues.

where the duration can be customized. The table below the word cloud presents the statistics of words in the collection. By clicking one emerging word, they can view the change history in line chart (*e.g.*, Fig. 3) and most relevant user comments for reference (*e.g.*, Fig. 1). To help developers get an overview of all the emerging issues, we also provide a list of emerging issues along with the clustered emerging word collocations and the corresponding prioritized user comments.

V. Evaluation Setup

A. Research Questions

To evaluate our tool, we set up some research questions to guide our evaluation. The research questions are as follows:

- ***RQ1:*** How effective is DIVER in detecting emerging app issues based on user feedback analysis?
- ***RQ2:*** How good is the quality of clustered word collocations for each emerging issue? Are their semantic meanings consistent enough for developers' understanding?
- ***RQ3:*** How efficient is DIVER in detecting emerging app issues in practice?

The baseline methods compared against DIVER to answer the above three research questions include: 1) the original IDEA method [14]; and 2) DIVER involving only the version dimension, denoted as DIVER^{-T}.

B. Datasets and Ground Truth

We deployed DIVER to the WeChat apps for Android and iOS. During the deployment, the apps were updated according to the emerging issues identified by DIVER. Meanwhile, we recorded the false and missing critical issues that were confirmed by WeChat senior developers. DIVER has been deployed to WeChat for more than one year and is still being used to monitor the emerging app issues. To measure the performance of DIVER, we take the total of 181,679 user reviews in early 2018 as our evaluation dataset. Details of these reviews are shown in Table I. Each user review has four attributes, including user identifier, operating system, post time, and review texts. We use the 18 emerging issues

identified by WeChat senior developers (using our tool and their own manual investigation) in early 2018 as our ground truth.

C. Evaluation Metrics

To answer *RQ1*, we adopt *Precision*, *Recall*, and *F-measure* as metrics to measure the effectiveness of DIVER, which are defined as below:

$$\begin{gathered} Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN} \\ F-measure = \frac{2*Precision*Recall}{Precision+Recall} \end{gathered} \tag{4}$$

where TP (true positive) indicates the number of issues that are correctly recognized as emerging issues, and FP (false positive) means the number of issues that are fasely identified as emerging issues. FN (false negative) is the number of emerging issues that are not identified by DIVER. The metric values range from 0 to 1, and a higher value signifies that a better performance is achieved.

To answer *RQ2*, we utilize the typical Normalized Pointwise Mutual Information (NPMI) metric [41] as the indicator to evaluate the semantic coherence of the words $\{w_1, w_2, \cdots, w_n\}$ (where n is total number of the words) for describing one emerging issue.

$$NPMI(w_i) = \sum_{j}^{T-1} \frac{\frac{\log P(w_i,w_j)}{P(w_i)P(w_j)}}{-\log P(w_i,w_j)}, \tag{5}$$

where the probabilities are derived from a 10-word sliding window over an external corpus following the standard [41]. A higher NPMI score indicates that the words in one emerging issue are more semantically coherent and easier for understanding.

VI. Results and Analysis

A. Answer to RQ1: Effectiveness of DIVER

Fig. 8 and Table II illustrate the results of *Precision*, *Recall*, and *F-measure* for the benchmarks and proposed tool.

1) DIVER vs. IDEA: Comparing DIVER with IDEA, we discover that DIVER consistently outperforms IDEA on all the datasets (as shown in Fig. 8), increasing *Precision*, *Recall*, and *F-measure* by 29.4%, 32.5%, and 29.5% respectively. Focusing on the metric *F-measure*, DIVER achieves 60.0%-94.9% scores on the subject apps, while IDEA's performance is only at 41.3%-54.5%. For WeChat iOS, which has a relatively smaller number of reviews among the two subject apps, DIVER increases the performance of IDEA by 5.5%. The results indicate that DIVER can better detect emerging app issues. We also find that DIVER can capture all the emerging issues in WeChat (including the Android and iOS apps), with recall at 100%. This mean that DIVER can help WeChat developers detect all important issues.

We further adopt analysis of variance (ANOVA) [42], a statistical hypothesis test for significance analysis, to examine whether DIVER can significantly outperform the benchmark method IDEA. We run both DIVER and IDEA on the subject apps for three times, and compute the p-value of ANOVA on the *Precision* results at 0.005. Sine the p-value is much less than 0.05, we obtain that DIVER can significantly better detect precise emerging issues than IDEA.

2) DIVER vs. DIVER^{-T}: We compare DIVER with DIVER^{-T} to observe whether involving history information from both time and version dimensions benefits emerging issue detection. As shown in Table II and Fig. 8, DIVER consistently achieves better average performance on the subject apps, with improvements in *Precision*, *Recall*, and *F-measure* at 4.2%, 0%, and 4.1%, respectively. Without considering the time dimension, DIVER^{-T} may even achieve poorer performance than IDEA, *e.g.*, for the WeChat iOS app. Focusing on the metric *Precision*, we find that DIVER^{-T} achieves inferior results on the subject app with relatively fewer versions (*i.e.*, WeChat iOS). This may be because that DIVER^{-T} only considers the version dimension, and the small number of versions makes it harder for DIVER^{-T} to learn to differentiate between true emerging issues and false positives. Thus, considering both time and version dimensions is helpful for accurately identifying emerging app issues.

B. Answer to RQ2: Word Collocation Quality Validation

We use the semantic consistency of the words in one emerging issue to access the comprehensibility of the issue. We adopt NPMI, introduced in V-C for evaluation. Table III illustrates median and average NPMI scores computed for the results of DIVER and IDEA. As shown in Table III, the words generated by DIVER for describing one emerging issue are more semantically coherent than those obtained from IDEA, with an average increase of NPMI scores of 2.15 times. According to [43], the word semantics in one topic are much coherent with NPMI score larger than 0.12. This indicates that the words for describing one emerging issue in DIVER (with avg. NPMI at 0.190) are semantically consistent.

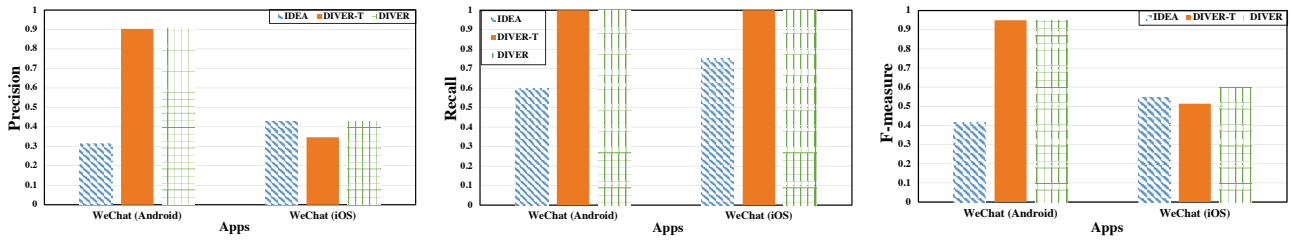
Table IV presents two example issues produced by DIVER and IDEA respectively. We discover that although using phrases (*i.e.*, more than one word) to describe issues may be favorable for issue understanding [14], the semantic consistency of these phrases is not guaranteed. For example, the "*sent by friend*" and "*unread message*" are not related to the emerging issue "*cannot hear sound*". Such semantic inconsistency will confuse developers if phrases in one emerging issue convey several meanings. Overall, the words generated by DIVER are more understandable for developers, which has also been confirmed by developers in WeChat.

C. Answer to RQ3: Efficiency of DIVER

To evaluate the efficiency of DIVER in detecting emerging issues, we measure the execution time of DIVER on the subject apps, and compare it with IDEA. For illustration, we randomly select an increasing percentage of 1,000 reviews from the WeChat Android dataset, and run both DIVER and IDEA until all the 1,000 reviews are selected. We run on a PC with Intel(R) Xeon E5-2620v2 CPU (2.10 GHz, 6 cores) and 16GB RAM. Fig. 9 displays the comparisons of

TABLE I: Experimental Datasets.

App Name	Category	Platform	#Ratings	Time Period	#Reviews	#Versions	#Avg. Reviews
WeChat	Communication	Google Play	5,313,722	Jan. 2018∼Feb. 2018	155,883	16	9,742
WeChat	Social Networking	App Store	38,497	Jan. 2018∼Feb. 2018	25,796	5	5,159



(a) Comparison on *Precision*. (b) Comparison on *Recall*. (c) Comparison on *F-measure*.

Fig. 8: Answer to *RQ1*: Effectiveness of DIVER.

TABLE II: Average results of the effectiveness of DIVER.

Framework	Precision	Recall	F-measure
IDEA	0.372	0.675	0.479
DIVER^{-T}	0.624	**1**	0.731
DIVER	**0.665**	**1**	**0.774**

TABLE III: Comparison of semantic consistency of words.

Metric	IDEA	DIVER
Med. NPMI	0.045	**0.139**
Avg. NPMI	0.059	**0.190**

time consumed on different data sizes for the two methods. According to Fig. 9, the time consumed by DIVER can be less influenced by the data sizes, while IDEA shows a dramatic increase on the time spent as the data size increases. We also find that DIVER can process 1,000 reviews in less than 10 seconds. Therefore, DIVER is much more efficient to detect emerging app issues and is more applicable for industry scenario where every second counts.

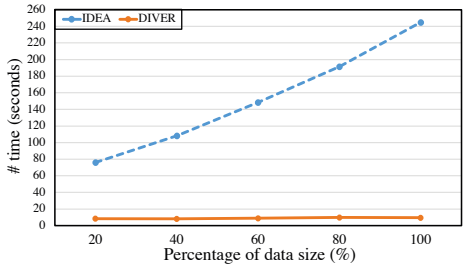


Fig. 9: Efficiency of DIVER and IDEA on different data sizes of 1,000 reviews.

VII. Qualitative Analysis

In this section, we present two successful cases and some failing ones of DIVER, from which we can further understand and identify how we can improve the performance of DIVER.

TABLE IV: Terms generated by IDEA and DIVER for the issues "*Cannot hear sound*" and "*Chatting records return too slow*" respectively. Red underlined fonts highlight the terms that are not semantically related to the issue topic.

Cannot hear sound (WeChat Android)		*Chatting records return too slow* (WeChat iOS)	
IDEA	DIVER	IDEA	DIVER
sent by friend	sound	full chatting records	fix
cannot hear sound	hear	publish content	slow
unread message	cannot	customer service	chatting records
-	voice	-	-

A. Successful Case I

Here, we demonstrate an emerging issue that was initially missed by developers, and could have been successfully detected if DIVER had been deployed. This is the issue highlighted in Section II-B. Specifically, DIVER could capture the issue "*cannot hear sound*" from the user feedback of an earlier beta test version of version X, published on June 30, 2017. The numbers of related user feedback between June 30 to July 7, and the corresponding proportions (comparing to all user feedback) are shown in Fig. 10. Although only 10 users complained about this issue on July 1, these reviews accounted for more than 30% of the whole feedback, and DIVER can identify it as an emerging issue. Thus, if DIVER were used at that time, the developers could detect the emerging issue effectively and in a timely manner.

B. Successful Case II

Another successful case of DIVER is the red packet issue occurred in WeChat iOS app. Specifically, on January 17, 2018, DIVER detected an emerging issue about "*send red packets*" in WeChat app for iOS, with the number and proportion changes of associated feedback shown in Fig. 11. According to the prioritized reviews relevant to the issue, developers quickly diagnosed the problem and localized the root case, *i.e.*, the backend payment environment in the testing version. DIVER enabled the development team to quickly

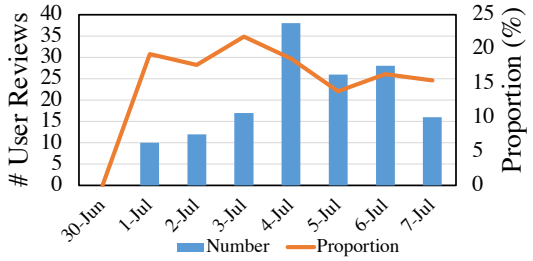
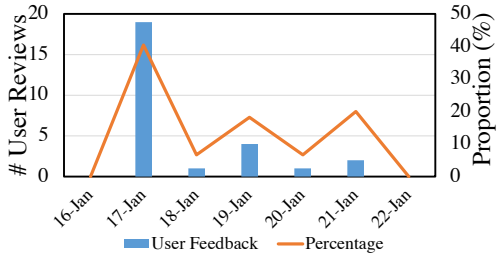


Fig. 10: Number and proportion changes of reviews related to "*cannot hear sound*" for one testing version of version X. June 30 is the release date of the testing version.

pinpoint and fix this emerging issue, thus reducing the cost for manually handling complaints and improving customer satisfaction.



(a) Number and proportion changes of reviews related to "*send red packets*" for one testing version of WeChat iOS. January 16 is the release date of that testing version.

Related Feedback:
① Can't send red packets. What's wrong with this? (from user_1)
② Don't know what happened. Can't send red packets. (from user_2)
③ Can't send red packets. What's the matter? (from user_3)
④ Can't send red packets and transfer money. (from user_4)
⑤ Can't send red packets. Why do we have to mark? (from user_5)

(b) Prioritized reviews related to "*send red packets*".

Fig. 11: The emerging issue "*send red packets*" detected in WeChat iOS app.

C. Error Case Analysis

We have also analyzed the error cases generated by DIVER. We find that most of the error cases (50%) are caused by non-informative user reviews. For example, DIVER detected one issue related to "*greeting people nearby*" from WeChat Android in January 2018, which corresponds to 18 user reviews. We illustrate the prioritized reviews of the issue in Fig. 12. This issue is later considered as a false positive by developers, since receiving replies from people nearby is not guaranteed for users sending out a greeting message. Such feedback is usually written by idle users, and cannot provide developers actionable information for app development.

Other error cases are either caused by non-informative single words that display significant increase (25%) or by the lower alarm threshold used during word collocation extraction (25%). Such cases can be alleviated by trying different

Keywords: Greet, Receive, See
Feedback No.: 18
Sample Related Feedback:
① People nearby can't see me. I greet them with a message but nobody seems to receive it. Please help to solve this problem. (from user_1)
② Greet nearby people but can't receive their replies. (from user_2)

Fig. 12: Prioritized reviews related to "*greeting people nearby*" for WeChat Android app.

anomaly threshold for single words and word collocations. Note that all emerging issue detection methods require the definition of thresholds, and such errors are not unique to our proposed tool.

VIII. Threats to Validity

Experimental Subjects: We select two industry apps for verification, which represent a small portion of the whole app markets and may lead to biased results. Such threat is unavoidable due to limited available apps with manually-labeled ground truth. In practice, DIVER has been deployed to monitor many other apps in Tencent, such as QQ music and Tencent browser[6], and its performance has been approved by the developers of these apps.

Parameter Influence: Similar to the topic modeling method, whose performance can be impacted by its predefined topic number, *etc.*, DIVER can also be affected by its hyper-parameters, including the window size and the threshold for determining emerging issues. Different from the methods based on topics, whose results can be greatly impacted by the random initialization of topic modeling, the outputs of DIVER are much more stable. Also, the hyper-parameters of DIVER can be estimated more easily in practice.

IX. Conclusion

For popular apps, app developers need to release and update their apps with fewer bugs and unsatisfactory features. For those apps, detecting emerging issues with delay would possibly adversely impact their users' experience and cause customer churn. Thus, we believe it is crucial to identify emerging app issues timely and accurately.

To address this need we have proposed a tool named DIVER and deployed it within Tencent. The proposed tool DIVER exploits the attributes (*e.g.*, post time and version) of user feedback, and conducts time-series analysis to detect emerging app issues. DIVER only needs to record the frequencies of app issues, which means that DIVER is easy to deploy for a new app. Also, experiments verify the effectiveness and efficiency of DIVER in emerging issue detection. As future work, we plan to conduct more industrial studies on user feedback analysis.

Acknowledgment

This work was fully supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14210717 of the General Research Fund).

[6]These apps serve hundreds of millions of users worldwide.

REFERENCES

[1] "Number of apps in app market," https://bit.ly/2dycQpS.
[2] "Pokemango got a lot of bad reviews," https://bit.ly/2MWuv81.
[3] "Wechat monthly active users," https://bit.ly/2DTAOos.
[4] C. Sharma, S. Sabharwal, and R. Sibal, "A survey on software testing techniques using genetic algorithm," *CoRR*, vol. abs/1411.1154, 2014.
[5] N. Leicht, I. Blohm, and J. M. Leimeister, "Leveraging the power of the crowd for software testing," *IEEE Software*, vol. 34, no. 2, pp. 62–69, 2017.
[6] T. Xie, "Transferring software testing tools to practice," in *12th IEEE/ACM International Workshop on Automation of Software Testing, AST@ICSE 2017, Buenos Aires, Argentina, May 20-21, 2017*, 2017, p. 8.
[7] H. Zheng, D. Li, B. Liang, X. Zeng, W. Zheng, Y. Deng, W. Lam, W. Yang, and T. Xie, "Automated test input generation for android: Towards getting there in an industrial case," in *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017*, 2017, pp. 253–262.
[8] S. McIlroy, N. Ali, and A. E. Hassan, "Fresh apps: an empirical study of frequently-updated mobile apps in the google play store," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1346–1370, 2016.
[9] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proceedings of the 21st IEEE International Conference on Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 125–134.
[10] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. D. Penta, "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*, 2016, pp. 14–24.
[11] F. Palomba, M. L. Vásquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "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*, 2015, pp. 291–300.
[12] M. Nayebi, B. Adams, and G. Ruhe, "Release practices for mobile apps - what do users and developers think?" in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, 2016, pp. 552–562.
[13] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "Ar-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. ACM, 2014, pp. 767–778.
[14] C. Gao, J. Zeng, M. R. Lyu, and I. King, "Online app review analysis for identifying emerging issues," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 48–58.
[15] A. Agrawal, W. Fu, and T. Menzies, "What is wrong with topic modeling? (and how to fix it using search-based SE)," *CoRR*, vol. abs/1608.08176, 2016.
[16] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *ICSE*. IEEE Computer Society, 2013, pp. 522–531.
[17] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Trans. Software Eng.*, vol. 43, no. 9, pp. 817–847, 2017.
[18] C. Iacob, V. Veerappa, and R. Harrison, "What are you complaining about?: a study of online reviews of mobile applications," in *Proceedings of the 27th International BCS Human Computer Interaction Conference (BCSHCI)*. British Computer Society, 2013, p. 29.
[19] H. Khalid, E. Shihab, M. Nagappan, and A. Hassan, "What do mobile app users complain about? a study on free ios apps," *IEEE Software*, vol. 10, 2015.
[20] E. Platzer, "Opportunities of automated motive-based user review analysis in the context of mobile app acceptance," in *Proceedings of the 2011 Central European Conference on Information and Intelligent Systems (CECIIS)*, 2011.
[21] Y. Man, C. Gao, M. R. Lyu, and J. Jiang, "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*, 2016, pp. 138–149.
[22] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *Proceedings of the 23rd International Conference on Requirements Engineering (RE)*. IEEE, 2015, pp. 116–125.
[23] F. Palomba, M. L. Vásquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "Crowdsourcing user reviews to support the evolution of mobile apps," *Journal of Systems and Software*, vol. 137, pp. 143–162, 2018.
[24] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 24th SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2016, pp. 499–510.
[25] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, 2013, pp. 41–44.
[26] E. Guzman and W. Maalej, "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*, 2014, pp. 153–162.
[27] X. Gu and S. Kim, ""what parts of your apps are loved by users?" (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 760–770.
[28] C. Gao, B. Wang, P. He, J. Zhu, Y. Zhou, and M. R. Lyu, "Paid: Prioritizing app issues for developers by tracking user reviews over versions," in *Proceedings of the 26th International Symposium on on Software Reliability Engineering (ISSRE)*. IEEE, 2015, pp. 35–45.
[29] W. Martin, F. Sarro, and M. Harman, "Causal impact analysis for app releases in google play," in *SIGSOFT FSE*. ACM, 2016, pp. 435–446.
[30] C. Gao, H. Xu, J. Hu, and Y. Zhou, "Ar-tracker: Track the dynamics of mobile apps via user review mining," in *2015 IEEE Symposium on Service-Oriented System Engineering, SOSE 2015, San Francisco Bay, CA, USA, March 30 - April 3, 2015*, 2015, pp. 284–290.
[31] B. Fu, J. Lin, L. Li, C. Faloutsos, J. I. Hong, and N. M. Sadeh, "Why people hate your app: making sense of user feedback in a mobile app store," in *KDD*. ACM, 2013, pp. 1276–1284.
[32] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," in *Proceedings of the 30th International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 749–759.
[33] P. M. Vu, H. V. Pham, T. T. Nguyen, and T. T. Nguyen, "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*, 2016, pp. 726–731.
[34] J. Zeng, J. Li, Y. Song, C. Gao, M. R. Lyu, and I. King, "Topic memory networks for short text classification," *CoRR*, vol. abs/1809.03664, 2018.
[35] "Jieba toolkit," https://github.com/fxsjy/jieba.
[36] M. Danilevsky, C. Wang, N. Desai, X. Ren, J. Guo, and J. Han, "Automatic construction and ranking of topical keyphrases on collections of short documents," in *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, 2014, pp. 398–406.
[37] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, 1994, pp. 487–499.
[38] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.
[39] "Gaussian distribution z table," https://bit.ly/2Ru7eeH.
[40] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008.
[41] J. H. Lau, D. Newman, and T. Baldwin, "Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality," in *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2014, April 26-30, 2014, Gothenburg, Sweden*, 2014, pp. 530–539.
[42] "Anova," https://en.wikipedia.org/wiki/Analysis_of_variance.
[43] Y. Miao, E. Grefenstette, and P. Blunsom, "Discovering discrete latent topics with neural variational inference," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 2410–2419.