User Review Mining for Assisting App Development

GAO, Cuiyun

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
September 2018

Thesis Assessment Committee

Professor CHAN Lai Wan (Chair)

Professor LYU Rung Tsong Michael (Thesis Supervisor)

Professor SUN Hanqiu (Committee Member)

Professor CHAN Wing Kwong (External Examiner)

Abstract of thesis entitled:

User Review Mining for Assisting App Development

Submitted by GAO, Cuiyun

for the degree of Doctor of Philosophy

at The Chinese University of Hong Kong in September 2018

Mobile app developers design user-friendly applications (apps) and update their apps to ensure good users' experience. User reviews serve as an essential channel between app developers and users, and deliver users' recent experience with the apps. By analyzing app reviews, developers can gain valuable information for app updates, including the features to improve, new functionalities sought-after by users, and also functional and non-functional issues to be rectified. Traditional review mining methods are mostly focusing on automatically analyzing a static review collection and provide little support for contrasting reviews across multiple time periods and dimensions. In this thesis, we propose automated review analysis methods for identifying important app issues over different time periods, along with evolving app versions, and across different app markets.

Firstly, we propose an aspect-tracking method for tracking the changes in user-concerned aspects from top-ranked reviews. Topic extraction from user reviews is one typical step to understand the delivered user experience. The extracted topics usually represent user-concerned app aspects. Instead of analyzing one static review collection, we explore the quantitative changes of prioritized topics along with time periods. By comparing different topic modeling methods, we propose effective topic-ranking and review-ranking schemes for capturing the most up-to-date issues. In this way, the

non-informative reviews can be avoided during the ranking. We further show a case study to indicate the effectiveness of tracking topic changes over time to promptly identify important issues.

Secondly, we propose a phrase-based issue prioritization method by analyzing reviews over different release versions. The method design is based on the fact that app issues presented in the level of phrase, *i.e.*, a couple of consecutive words, can be more efficiently understood by developers than those presented in long review sentences. We automatically label ranked topics with most representative phrases based on their semantic relevance, and visualize the topic trends with ThemeRiver for facilitating developers' observation. App changelogs, which record the changes of current release, are utilized to measure the effectiveness of our method in detecting important app issues for app evolution.

Thirdly, we design an automated online review analysis framework for identifying emerging app issues. Emerging issues are the issues that rarely appear in previous app versions, but occupy significant proportions of user reviews in current versions. Detecting emerging issues timely and precisely offers great help to developers in updating their apps. We propose an online topic modeling method for generating version-sensitive topic distributions, and employ typical anomaly detection method to determine the emerging topics. The topics are automatically interpreted with the prioritized phrases and review sentences based on an effective ranking scheme considering both semantic relevance and user sentiment.

Fourthly, we conduct an empirical study to explore whether there exist significant differences among app issues on three app markets, *i.e.*, Google Play, Apple's App Store, and Windows Store. Understanding differences of app issues on these platforms can assist developers in efficiently choosing corresponding testing cases. We focus on analyzing seven general app issues, including battery, crash, memory, network, privacy, spam, and UI. We propose an issue-related keyword extraction method for establishing a keyword

dictionary for each app issue. An extensive study on around five million user reviews of 20 apps shows issue distributions of apps designed for different app markets exhibit significant difference.

Lastly, we focus on studying an important component for many mobile apps, *i.e.*, in-app advertising (ad). Mobile resource occupation caused by in-app ads is non-negligible for app developers to ensure a good user experience and continuous profits. Thus, we explore the effects of in-app ads on user experience by analyzing app reviews from Google Play. We prioritize concrete user concerns about in-app ads by mining ad-related user feedback, measure ads' practical performance costs, and observe user opinions on the performance costs of ads in practice. We obtain six insightful suggestions on designing user-friendly in-app ads.

In summary, the thesis targets at user review analysis for facilitating app developers in the app design, testing, and updating processes. Large-scale experiments on real-world datasets demonstrate the effectiveness of our proposed methods in various applications.

論文題目：以輔助應用程式開發的用戶評論分析方法

作者　　：高翠芸

學校　　：香港中文大學

學系　　：計算機科學與工程學系

修讀學位：哲學博士

摘要　　：

手機應用程式開發者為確保良好的用戶體驗需要設計用戶友好型的應用，並對其及時更新。用戶評論作為開發者和用戶之間的主要媒介，傳達了最近的用戶體驗。通過分析用戶評論，開發者可以獲得有助於應用更新的有效信息，包含需要完善的特徵，用戶需求的新功能，以及待糾正的功能性和非功能性問題。傳統的評論挖掘方法大多集中在自動化分析靜態的評論數據集，且提供較少的多時間和多維度的對比。在本論文中，我們提出了跨時間、跨版本、跨平台的自動化分析方法以識別重要的應用問題。

首先，我們提出了基於評論排序的主題追蹤方法。從用戶評論中提取主題是理解用戶體驗的很經典的步驟。提取的主題通常代表了用戶對應用所關心的方面。區別於以往的基於靜態評論數據集的研究，我們探索了隨著時間變化優先主題的

數量變化。通過比較不同的主題模型方法，我們提出了有效的主題排序和評論排序方法。同時，無信息含量的評論也在排序的過程中被過濾掉。通過案例研究，我們進一步展示了追蹤主題隨時間變化對快速發現重要問題的有效性。

其次，我們通過分析連續版本的用戶評論，提出了基於詞組的問題優先化方法。該方法設計是基於用詞組表示的應用問題比用長的評論句子表示的應用問題能夠更高效地被開發者所理解的事實。我們自動地對為每個主題用最具有代表性的詞組打標簽，並用主題河來可視化主題趨勢以輔助開發者觀察。記錄了當前版本的應用更新日誌被用來驗證我們方法在檢測重要應用問題的有效性。

再次，我們設計了在綫評論自動化分析框架來發現突發的應用問題。突發應用問題是指在之前的應用版本中比較少出現，而在當前版本的用戶評論中比例顯著。及時並準確地檢測突發問題對開發者更新應用程式提供了非常極大的幫助。其中我們提出了在綫主題模型方法來生成對版本敏感的主題分佈，然後採用經典的異常檢測方法來確定突發主題。最後，基於兼顧語義相關性和用戶情感的有效的排序機制，這些主題被自動地用優先性最高的詞組和評論句子來解釋。

然後，我們執行了經驗研究來探索不同應用市場的應用問題

是否存在顯著差異。這些應用市場包括谷歌市場，蘋果應用商店和微軟應用商店。理解這些平台上應用問題的差異可以幫助開發者高效地選擇相應的測試用例。我們集中分析七種常見的應用問題，包含電量、崩潰、內存、網絡、隱私、垃圾信息以及界面。我們為每個應用問題簡歷一個關鍵詞詞典，並提出了提取於問題相關的關鍵詞的自動化方法。基於 20 個應用程式的將近五百萬條評論上的實驗表明不同應用市場上的應用問題分佈存在顯著性差異。

最後，我們集中分析許多免費移動應用程式中很重要的組成部分，即應用內廣告。由應用內廣告產生的手機資源消耗對開發者確保良好的用戶體驗和持續的廣告利潤來說不可忽視。因此，我們通過分析谷歌市場上的用戶評論探索了應用內廣告對用戶體驗的影響。通過分析與廣告相關的用戶評論，我們對用戶對於廣告的具體的關心問題進行排序，測量了廣告的實際性能消耗，並觀察用戶是否對實際的性能消耗敏感。我們獲取了六種有助於開發者設計友好型應用內廣告的建議。

綜上所述，本論文的目標是通過用戶評論分析，來幫助應用程式開發者的應用設計、測試以及更新環節。大量基於真實數據的實驗驗證了我們提出的方法在不同應用中的有效性。

Acknowledgement

I feel highly privileged to take this opportunity to express my sincere gratitude to the people who have been instrumental and helpful on my way to pursuing my Ph.D. degree.

First and foremost, I would like to thank my supervisor, Prof. Michael R. Lyu, for his kind supervision of my Ph.D. study at CUHK. He has provided inspiring guidance and incredible help in every aspect of my research. From choosing a research topic to working on a project, from technical writing to paper presentation, I have learned so much from him not only on knowledge but also on attitude in doing research. Besides, he gives me a lot of freedom to select my research topic and study necessary techniques. I will always be grateful for his advice, encouragement, and support at all levels.

I am grateful to my thesis assessment committee members, Prof. Lai Wan Chan and Prof. Han Qiu Sun, for their constructive comments and valuable suggestions to this thesis and all my term reports. Great thanks to Prof. Wing-Kwong Chan from the City University of Hong Kong who kindly served as the external examiner for this thesis.

I would like to greatly thank my oversea supervisors, Prof. Mark Harman and Dr. Federica Sarro, for their support of my visit to University College London. During the visit, Prof. Harman and Dr. Sarro had provided insightful ideas and constructive feedback for my research. I also greatly thank Jie Zhang, Bill Langdon, Hector D Menendez, Afnan A. Al-Subaihin, Bobby Bruce, Carlos Gavidia,

Chaiyong Ragkhitwetsagul, DongGyun Han, Matheus Henrique Esteves Paixao, Profir-Petru Partachi, and Iason Papapanagiotakis-Bousy for the three-month wonderful memories in UCL.

I would like to thank Dr. Chin-Yew Lin, my mentor during the internship at Microsoft Research Asia. I also really thank friends met in MSRA, Jinpeng Wang, Jing Liu, Feng Nie, Danqing Huang, Longxu Dou, and Xinya Du, for their kindness and help.

I would like to thank Dr. Wujie Zheng, my mentor during the collaboration in WeChat of Tencent. I also thank friends met in Tencent, Yuetang Deng, Dian Liu, Jie Xiong, Jiayang Tu, and Zhao Chen, for their support and encouragement.

I would like to thank Dr. Chao Zhou, my mentor during the internship in Baidu. I also thank friends met in Baidu, Guang Lin, Bin Wu, Jiaxin Lin, and Yan Zhang, for their help and inspiration.

I would like to thank my life-long friends, Xuemei Liu, Wenjing Gao, Yiwei Li, Wen Liu, Rong Yang, Jing Xing, Xiaoxuan Ou and Ping Jiang, for their trust and patience.

I thank Yangfan Zhou, David Lo, Yichuan Man, Hui Xu, Jieming Zhu, Junjie Hu, Baoxiang Wang, Chin-Yew Lin, Federica Sarro, and Pinjia He, for their valuable guidance and contribution to the research work in this thesis. I am also thankful to my other groupmates, Zibin Zheng, Haiqin Yang, Jian Li, Shilin He, Yilei Zhang, Chen Cheng, Yu Kang, Tong Zhao, Hongyi Zhang, Shenglin Zhao, Xixian Chen, Yuxin Su, Xiaotian Yu, Pengpeng Liu, Yue Wang, Wenxiang Jiao, Jingjing Li, Yifan Gao, Weibin Wu, and Zhuangbin Chen, who gave me encouragement and kind help.

Last but not least, I would like to thank significantly my parents, and my husband and also groupmate, Jichuan Zeng. Without their everlasting love and patience, this thesis would never have been completed.

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents our research towards user review analysis for assisting mobile app developers, which is currently an important field of study and practice in app evolution, testing, and maintenance. We provide a brief overview of the research problems under study in Section 1.1, and highlight the main contributions of this thesis in Section 1.2. Section1.3 outlines the thesis structure.

1.1 Overview

Smartphones penetrate many facets of everyday life, including entertainment (*e.g.*, game playing), connectivity with others (*e.g.*, social networking), and efficient work (*e.g.*, file sharing). By 2018, over 36% of the world's population is projected to use a smartphone, up from about 10% in 2011 [123]. Google's Android system and Apple's iOS system are the most two popular smartphone operating systems in the industry. To enjoy the convenience provided by mobile phones, users need to download and use mobile apps. The major distribution channel for mobile apps is an app store, such as Google Play, Apple's App Store, and Windows Phone Store. With more and more user consumption on mobile apps, more developers are motivated to design and publish apps on these app stores by the potential revenue (*e.g.*, mobile ads and in-app purchase) and

relatively easy implementation [41]. The number of apps in app stores is thereby increasingly growing. For example, Google Play and App Store provide around 3.8 million and two million apps in 2018, respectively. To make their apps outstanding among the vast numbers of apps, developers should design their apps with attractive or important functionalities, and most importantly, ensure good user experience.

Different from traditional software repositories such as source code [80] and API documents [133], user reviews on mobile apps are direct and instant feedback from the users who have experienced the apps. They reflect the practical user experience, in terms of bug fixing, feature refinement, and functionality request. Analyzing app reviews provides developers an opportunity to proactively collect these user complaints, and facilitate scheduling of next releases, as shown in Figure 1.1.

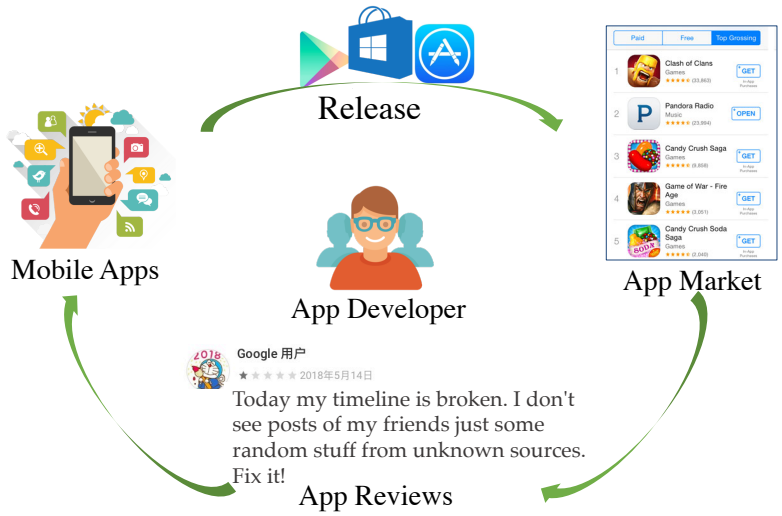


Figure 1.1: App development cycle.

The characteristics of user reviews make efficient and effective review analysis very challenging. First, app reviews are generated every day in large volume. Manual analysis is prohibitively time-consuming for apps with large numbers of reviews (*e.g.*, Facebook receives more than 10,000 reviews in Google Play every day [2]).

Second, app reviews contain numerous noise words, such as misspelled words, repetitive words, non-English words, and casual words. Also, they are often shorter in length and limited in context, since most of them are written by users via mobile terminals. Third, only 30% of the reviews provide informative user opinions for app updates [52]. Filtering out non-informative reviews manually is non-trivial and labor-intensive. Furthermore, detailed issues specific to each app are hard to be predefined, because they are diverse for different apps and versions. All these characteristics lead automatic review analysis to be a non-trivial task.

Previous research mainly focuses on reducing the manual power in extracting software aspects or user preferences, such as establishing dictionaries for preprocessing reviews and assisting review retrieval [186], filtering out non-informative reviews [52], or classifying reviews to predefined topics [167]. Some research work [72, 76] is devoted to predicting user sentiment about specific app features, which can be used for app maintenance. These studies generally act on static review collection, and do not consider the timeliness feature of apps. However, apps are typically rapidly-evolving software, and app development is incremental and iterative. This signifies that app reviews are continually updating along with new versions released, and the time-sensitive characteristic of reviews is helpful and non-negligible for comprehensive review analysis.

Better app evolution (or release planning) [132] based on user reviews has drawn more and more attention recently. Release planning of mobile apps refers to making proper decisions about the functionalities of the evolving app releases. It is critical for the success or failure of an app [132]. In [96, 63, 86, 116], the variations of app ratings, prices, or reviews sizes over time are explored. To determine which issues to fix or improve, the previous work [142, 185, 141] generally relies on predefined topics, such as bug reporting, feature request, and GUI, etc. However, the newly-appearing and detailed app issues (*e.g.*, newly-added app features)

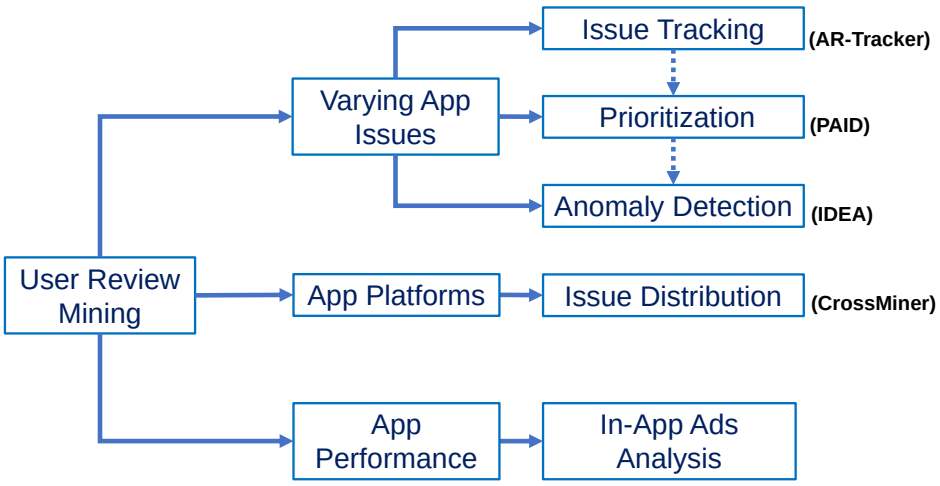


Figure 1.2: Overview of our user review mining framework.

are hard to be determined manually in advance for popular apps, and rarely studied by the previous work. The challenging point is that the topics are varying with different version releases, and they are diverse for different apps. In this thesis, we overcome this challenge, and detect the detailed issues over time based on prioritized reviews and top-ranked phrases with reduced manpower.

The research of this thesis comprises five parts. The overview of the user review mining framework is depicted in Fig. 1.2. It contains three major topics, including analysis on varying app issues, cross-platform understanding, and app performance exploration. During the exploration on ever-changing app issues (the first three parts of the thesis), we first show a case study illustrating the usefulness of tracking app issues (AR-Tracker), then a phrase-level issue tracing framework for prioritizing app issues along with version releases (PAID), and finally an emerging issue detection framework for identifying anomaly issues timely (IDEA).

The diversity of app platforms is another characteristic of app reviews. To make apps more publicly available, developers generally choose to deliver their apps on more than one platform to enlarge the potential user volume and revenue [27]. Due to

the differences in operating systems and user preferences of these platforms, user-concerned issues may exist distinctions for the same app. Understanding the issue distributions on different platforms can help developers invest limited energy in testing and modifying the issues that users care more about. In the fourth part of the thesis, we study the distributions of critical issues on the three major app platforms (*i.e.*, Google Play, App Store, and Windows Store), and summarize the similarities and differences on these platforms (CrossMiner).

Mobile Apps' performance, such as mobile resource occupation, has a vital impact on user experience and continuous profits [107]. In this thesis, we focus on exploring the performance cost of in-app advertising (ad), as in-app ads are the primary source of revenue for many free mobile apps. Previous work is mainly devoted to addressing performance cost generated by ads [182, 164], or resorting surveys to collect general factors that impact users' acceptance of ads [199, 168, 55]. However, users' detailed concerns about ads, and their attitude towards ads' practical performance cost (*e.g.*, memory cost) have rarely been studied. In the fifth part of the thesis, we determine concrete user concerns about in-app ads based on user review mining, and explore user opinions about the performance costs of ads in practice (*i.e.*, whether more performance costs can generate more user concerns.). Our exploration of in-app ads provides several actionable guidelines for in-app advertising.

1.2 Thesis Contributions

In this thesis, we make contributions to user review mining in the following ways:

1. **Issue Tracking based on Prioritized Reviews**

   App reviews are usually massive in size, mixed with non-informative reviews, and span over multiple issues, thus lead-

ing to great challenges for developers to efficiently track the quantitative changes of app issues, and identify the key reviews of interest. The existing methods require extensive human efforts to manually label informative reviews for filtering noisy reviews out, and generally focus on mining the static review collection. However, app reviews are updating as new versions come out, and they only reflect app issues of specific app versions. The dynamic information delivered by user reviews of different review periods can help developers determine the bugs to fix or features to add. To this end, this thesis develops AR-Tracker, a tool specifically designed for issue tracking based on prioritized reviews. We propose a review-ranking algorithm based on top-ranked topics, where reviews convey more important user-concerned topics would be prioritized. Experimental evaluation of four popular mobile apps with 500k user reviews illustrate that our tool can automatically surface informative and urgent reviews, comparing to the state-of-the-art method. We track the quantitative changes of app issues by manually checking the top-ranked reviews. We also show a case study which indicates that tracing user reviews along with time periods can assist developers in timely identifying important user demands.

2. **Issue Prioritization over Release Versions**
   User review analysis is critical to the bug-fixing and version-modification process for app developers. Manually tracing app issues based on prioritized reviews still consumes developers amounts of time in understanding user intentions in the top-ranked reviews. Also, hypothetically, in contrast with phrase-level issues, sentence-level app issues generally cost developers more time to grasp the important ones. In this thesis, we develop a framework called PAID for automatically prioritizing phrase-level app issues for each version. Besides, one major missing point of the previous work on issue tracking is that the

result verification is usually qualitative and not quantifiable. This impedes the generalization of the problem. To mitigate the limitation, we propose to employ official app changelogs as ground truth for verifying the performance of issue tracking. We aim at prioritizing the app issues that are considered important by developers during new version design. We conduct large-scale experiments on 18 popular apps with millions of reviews to verify the effectiveness of PAID.

3. **Emerging Issue Detection**
   Detecting emerging issues (*e.g.*, new bugs, features to improve, and functionalities to add) timely and precisely is critical for developers to update their apps. App reviews provide an opportunity to proactively collect user complaints and promptly improve apps' user experience, in terms of bug fixing and feature refinement. However, the tremendous quantities of reviews and noise words (*e.g.*, misspelled words) multiply the difficulties in accurately identifying newly-appearing or suddenly-increasing app issues. To address this challenge, in this thesis, we define these issues as emerging issues, and propose an automated framework called IDEA to identify the emerging app issues/topics effectively based on online review analysis. We evaluate IDEA on six popular apps from Google Play and Apple's App Store, employing the official app changelogs as our ground truth. Experiment results demonstrate the effectiveness of IDEA in identifying emerging app issues. Feedback from engineers and product managers shows that 88.9% of them think that the identified issues can facilitate app development in practice. Moreover, we have successfully applied IDEA to several products of Tencent, which serve hundreds of millions of users. The code and review data are open-source released online.

4. **Issue Understanding Across App Platforms**

App developers publish their apps on different app platforms, such as Google Play, App Store, and Windows Store, to maximize their user volumes and potential revenues. Apps for different platforms are particularly designed for specific operating systems. Due to the different characteristics of the operating systems and user preference on the platforms (*e.g.*, Android provides a more customizable platform, and iOS attaches more attention to the user interface.), app issues on these platforms may exhibit significant difference. Understanding the differences in app issues on the platforms can help developers prioritize their design and testing effort. In this thesis, we propose a framework named CrossMiner to automatically prioritize app issue and explore that whether the issues on the platforms are distributed differently. We focus on analyzing seven app issues, including battery, crash, memory, network, privacy, spam, and UI. Extensive experiments on around five millions of app reviews from 20 popular apps show that apps designed for the platforms present significantly different issue distributions. Source code and input data are also made publicly available.

5. **Exploration of the Effects of In-App Ads on User Experience**

   One primary revenue model adopted by many mobile apps is in-app ads that are embedded in the apps and displayed at various points during usage. The integration of ads has strong impacts for both users and app developers. For example, the consumption of ads on mobile resource and distraction to users' interactions with apps may cause customer churn, thus leading to reduced ad profits. Gaining in-depth knowledge of users' actual feelings when interacting with ads is helpful to developers for designing user-friendly ads. To this end, we conduct a comprehensive empirical study to explore concrete user concerns about in-app ads and user opinions on

the performance costs of ads in practice. We quantify user concern levels about specific ad issues based on user review analysis. Our work provides several actionable guidelines for in-app advertising, which have been endorsed by 94.6% of the engineers in a developer survey, such as shortening compulsory video ads, and embedding ad SDKs which support accurate ad recommendation.

1.3 Thesis Organization

- **Chapter 2**
  In this chapter, we review some background knowledge and related work on user review mining for assisting mobile app development. First, we introduce the typical topic modeling methods and the neural network method for short text understanding, as the methods are adopted or improved in our situations and throughout the thesis. Then we review previous work on app review analysis, including user intention mining, user sentiment prediction, assistance in release planning, and cross-platform study. Finally, we review the previous exploration on ad costs, and briefly clarify the difference comparing with our study.

- **Chapter 3**
  This chapter presents an automated tool for prioritized informative user reviews based on topic ranking, with the aim to trace app issues over different time periods. We show a case study that tracking app issues can assist developers in timely detecting app bugs to fix or features to improve. More specifically, in Section 3.1, we introduce the motivation and challenges of automatic extracting valuable information from user feedback. Section 3.2 presents the overview of our tool and detailed steps in the tool implementation. Section 3.3

elaborates on the subject dataset, comparison results on the performance of the proposed review ranking method, and the effectiveness of tracing app aspects reflected in user reviews. We conclude this chapter in Section 3.4.

- **Chapter 4**
  In this chapter, we propose an issue-ranking framework, namely PAID. PAID prioritizes phrase-level app issues along with release versions with minimal manual labor. We aim at capturing critical app issues for determining changes for next release. The official changelogs are chosen as ground truth for the verification. More specifically, Section 4.1 introduces the motivation for exploring the trends of concrete app features from user reviews. Section 4.2 illustrates the workflow of PAID, which includes three main steps, and its implementation details of each step. Section 4.3 explains the experimental dataset and the evaluation results. We discuss the validity and generalizability of PAID in Section 4.4, and summarize this chapter in Section 4.5.

- **Chapter 5**
  This chapter presents an emerging issue detection framework, namely IDEA, which can automatically identify emerging app issues based on online app review analysis. Although prioritizing issues can help developers schedule app modification, the emerging issues are relevantly more important and urgent. More specifically, Section 5.1 presents the motivation of emerging issue detection. The background of IDEA, including the concept of emerging app issues, the importance of online review analysis, and our ground truth (*i.e.* the app changelogs), is introduced in Section 5.2. Section 5.3 describes the overall framework of IDEA, which includes four components, and the details of each component. We evaluate the performance of IDEA based on case studies in Section 5.4. The practical

effectiveness of IDEA and the threats to validity are displayed in Section 5.5 and Section 5.6, respectively. Finally, we conclude this chapter in Section 5.7.

- **Chapter 6**
  In this chapter, we present an empirical study on exploring differences and similarities of issue distributions on different app platforms. We focus on studying three popular app platforms, including Google Play, App Store, and Windows Store, and seven app issues (*i.e.*, crash, battery drainage, memory consumption, network connection, privacy, spam, and UI design). We design an effective tool, namely CrossMiner, for retrieving keywords relevant to specific app issue. More specifically, we introduce the motivation of exploration on different app platforms in Section 6.1. Section 6.2 elaborates some background knowledge and detailed motivation. Section 6.3 gives an overview of our tool, CrossMiner, and explains the two major procedures in CrossMiner. Section 6.4 illustrates the performance of CrossMiner in issue prioritization and platform-level observations. We discuss the threat to validity in Section 6.5, and conclude this chapter in Section 6.6.

- **Chapter 7**
  This chapter explores the effects of in-app ads on user experience based on app reviews from Google Play. We study the major ad issues expressed by users and users' opinions on the performance costs of ads in practice. More specifically, Section 7.1 presents the motivation and challenges of our in-ads exploration. Section 7.2 details the methodology used during the exploration. We illustrate the subject datasets and insights obtained during experimental analysis in Section 7.3. One case study 7.4 is conducted to indicate the usefulness of the obtained insights. We finally conclude this chapter in Section 7.5.

- **Chapter 8**

The last chapter summarizes this thesis and provides some future directions that deserve further exploration.

To make each chapter self-contained, we may briefly reiterate the critical contents, such as motivations and framework descriptions, in some chapters.

□ **End of chapter.**

Chapter 2

Background Review

This chapter briefly reviews some background knowledge and related work of our research. App reviews are usually short in length as they are often written by users from mobile keyboards. First, we provide background knowledge about short text understanding in the field of natural language processing in Section 2.1, including topic modeling based methods and typical neural network methods. Then in Section 2.2, we introduce previous work on app review analysis in four main aspects, *i.e.*, user intention mining, user sentiment prediction, assistance in release planning, and cross-platform study. Finally, we review related studies on ad cost exploration and illustrate the difference of our research from them in Section 2.3.

2.1 Short Text Understanding

Short texts (such as tweets, web search snippets, news feeds, and forum messages), have become an important form for individuals to voice opinions and share information on online platforms. There is a growing demand for automatic language understanding techniques for processing and analyzing such content. The challenge of accurate short text understanding stems from the data sparsity issue. To alleviate such data sparsity, previous work improves topic modeling methods to better represent the topic distributions in the short text

collection, or modifies neural network methods for learning better word representations.

2.1.1 Topic Modeling Method

Topic modeling methods [45, 83] are designed to implicitly infer word co-occurrence patterns at document-level, to present topic structure. The key idea is to map high-dimensional count vectors, such as vector space representations of text documents [161], to a lower dimensional representation. Mining semantic structure in a text collection can be traced back to Latent Semantic Analysis (LSA) [58], which adopts the singular value decomposition of the document-term matrix to reveal the major associative word patterns. Based on topic modeling, we can obtain topic distributions of words and documents in the collection. Probabilistic Latent Semantic Analysis (pLSA) [83] is proposed to improve the (LSA) in a probabilistic sense by using a generative model. The pLSA can model documents as a mixture of multinomial distributions basing on a statistical latent variable model for factor analysis of count data. pLSA has been successful in many real-world applications, including computer vision, and recommend systems. However, since the number of parameters grows linearly with the number of documents, pLSA is confronted to a large number of estimation parameters. The offline nature of pLSA also makes it incapable to apply to unseen documents. Latent Dirichlet Allocation (LDA) [49] is a generative probabilistic model that represents a Bayesian upgrade to pLSA by introducing Dirichlet priors on document-topic distributions. LDA resolves problematic issues of pLSA such as increasing number of estimation parameters and inability to be applied incrementally to unseen documents by placing a Dirichlet prior distribution. LDA has an increased complexity than pLSA, several approximate inference algorithms are derived, such as Variational Inference [50], and various Markov Chain Monte Carlo algorithms, such as Gibbs

Sampling [70], can efficiently infer the model parameters. LDA is one of the most popular methods in topic modeling. There are lots of complex variations of LDA are proposed to model relationships between topics [47], to model evolution of topics over time [48], to model the hierarchical topics [46], to model authorship [155] and others.

With the emergence and prosperity of social media, topic models have been utilized for social media content analysis in various tasks, such as classification[170], content characterizing [151], event tracking [103], comment summarization[110], content recommendation [146], user interest profiling [192], and topic detection [189]. In early stage, some researchers directly applied conventional (or slightly modified) topic models for analysis [151, 37].

Generally, text collections with more word co-occurrence would generate more reliable topic inference. Thus, short texts, which contain fewer co-occurred words than long texts, are suffered a lot from the data sparsity problem in short texts, leading to inferior topic inferences with conventional topic models in topic inference [100]. Earlier studies focus on exploiting external knowledge to enrich the representation of short texts. In [145], Phan *et al.* propose to infer topic distribution of short texts by using the latent topics from Wikipedia. Similarly, the work [93] propose to infer latent topics of short texts for clustering by using auxiliary long texts. Sahami and Heilman [160] suggest a search-snippet-based similarity measure for short texts. These models require a large high-quality regular text corpus, which maybe domain and/or language specific. Many aggregation strategies have been proposed by merging short texts into long pseudo-documents to enrich context information in short texts. Conventional topic modeling is then applied to infer the latent topics. For example, Weng *et al.* [192] propose to aggregate tweets from the same user as a pseudo-document before performing the standard LDA model. Other metadata (e.g., timestamps, hashtags) that have been used for short text aggregation [84, 118]. However,

such aggregation metadata may not be available in some domains, e.g., news title and search snippets. These studies are empirical and hard to extend for general short texts topic modeling.

Topic modeling specific designed for general short text is studied in later studies. Yan *et al.* [195] propose a novel biterm topic model (BTM) to explicitly model the generation of word co-occurrence patterns instead of single words as do in conventional topic models. BTM is demonstrated to generate good discriminative topic representations as well as more coherent topics. In [197], the authors proposed to employ a simple and effective topic model, named Dirichlet Multinomial Mixture (DMM) model to discover latent topics in short texts. DMM is inspired by author-topic [155] model, in which each document has a single author, and based on the assumption made in the mixture of unigrams model proposed by [135], i.e., each document is sampled from a single latent topic. Such an assumption is suitable for short text because of the data sparsity issue. DMM is proven to be more effective than conventional topic models in many follow-up short text studies [201, 150, 99].

2.1.2 Neural Network Method

Learning word representations is an important step for understanding texts. To reduce the computation complexity of representation learning, Mikolov *et al.* [121] propose two shallow neural network architectures, *i.e.*, the Skip-gram model and the Continuous Bag-of-Words model. At the meantime, to handle the intractability of full softmax function at the output, several solutions have been proposed, either using hierarchical versions of softmax [127] or unnormalized models for training [122]. Among these variants of Skip-gram model, the Skipgram model with negative sampling [121] has achieved the state-of-the-art results in several evaluation tasks of word embeddings, including the analogy reasoning, sentiment analysis, sentence completion and so on.

These models use a shallow neural network with only one hidden layer to learn the relationship between the center word and the context words and obtains the hidden weights as word vectors. It is capable of learning semantic and syntactic meanings among words, and mapping similar words into nearby locations in the vector space. The simplicity enables it to train on huge datasets with billions of token within a short period. By arithmetic operations on word vectors, it is able to produce meaningful phrases, which is quite amazing.

As the neural network language models became popular, word embedding learned from external corpus are incorporated into topic model to help fix the limited content issue of short text. Qiang *et al.* [149] propose an embedding-based topic model (ETM) for short texts, inspired by the aforementioned aggregation strategies. ETM incorporates the word correlation knowledge provided by words embedding over the latent topic to cluster short texts to generate long pseudo-text. In the work [99], Li *et al.* propose GPU-DMM extends the DMM model by incorporating the learned word embedding (relatedness) from a large text corpus.

2.2 App Review Analysis

User feedback analysis has recently attracted the attention of software engineering researchers in several explorations, including app feature identification [139, 89, 77], prioritizing app issues [190, 64, 112, 65], and code localization [56, 140], etc. App reviews serve as a communication bridge between developers and users. They are provided by users with actual usage experience and are not biased by questionnaire specific issues [147]. Analyzing app reviews can help developers understanding user requests and existing app bugs in a timely manner. The observed information provides developers with clues about scheduling app modification and designing testing cases. However, automated and effective review mining is inherently

challenging due to the large quantity and noisy nature of user review data. We review the previous manual or automatic review mining tasks, in terms of mining user intention, predicting user sentiment, assisting app evolution, and cross-platform exploration.

2.2.1 User Intention Mining

Mining user intention aims at accurately capturing users' requests and purposes delivered by their reviews. Iacob *et al.* [90] 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.* [95] manually tag 6,390 low star-rating reviews from iOS apps and reach the conclusion that the most frequent complaints are related to functional errors, feature requests, and app crashes. Ciurumelea *et al.* [56] manually analyzed 1,566 user reviews and defined different levels of taxonomy contain mobile specific categories (such as performances and resources), based on which they recommend source code files for modification. Driven by the increasing amount and importance of user reviews, there exists research effort [147, 111, 139, 167] aiming at automating the review tagging process. For example, Iacob and Harrison [89] automatically extract the reviews related to feature requests based on predefined linguistic rules. Platzer [147] groups 16 types of basic desires according to usage motives that are addressed in the text, and uses multi-class classification method to predict the user motives. Chen *et al.* [52] focus on prioritizing the informative reviews by employing the typical topic modeling method LDA based on filtered reviews. Sorbo *et al.* [167] introduce two-level classification model to predict both user intention (*e.g.*, information giving and problem discovery) and review topic (*e.g.*, GUI and contents) of one review. Vu *et al.* [186] propose a keyword-based framework called MARK for semi-automated review analysis. MARK allows developers to

retrieve the most semantically relevant reviews to a set of interested keywords. Martin *et al.* [116] conduct a comprehensive survey on app store analysis, including app review mining.

2.2.2 User Sentiment Prediction

Generally, the sentiment analysis approaches can be categorized into two types, based on machine/deep learning [113, 98] or lexicons [98, 176]. The former approaches usually require manually-labeled data, which limits the generalization of trained models cross domains. Lexicon-based approaches are more flexible for distinct domains and involve much less labor, which is also the foundation of the feature sentiment analysis in our work, where the app feature can be regarded as one product aspect.

In software engineering area, identifying "what parts of software are used/loved by users" is one of the most important questions software developers care about [43]. Online app platforms such as Google Play and App Store provide a channel for users to express opinions and polarized sentiment (one to five stars) on experienced apps. High rantings and positive reviews usually promote the availability of an app. Although user reviews are generally connected with user ratings, users' attitude towards specific features is not easily achieved. Hoon *et al.* [85] analyze 8.7 million reviews on App Store and discover that the most frequently-used words in user reviews are likely to be sentiment words. They also find that the words describing negative opinions are significantly more than those expressing positive sentiment. Guzman and Maalej [76] propose to automatically score fine-grained app features with user sentiments, where the app features are grouped phrases by LDA. Evaluation on seven subject apps from the Apple App Store and Google Play Store verify the effectiveness of their approach. Gu and Kim [72] focus on capturing feature-opinion pairs with F1-score 0.81, which significantly outperform Guzmans' method (0.55). However, their

feature-opinion pairs are clustered based on common words in feature descriptions, may ignoring semantically similar but not same features, such as the feature "UI" and "interface". Thus the feature sentiment may be biased across app reviews. Luiz *et al.* [108] propose a general framework that allows developers to filter and analyze user sentiment about specific app features. Topic modeling method is adopted to extract semantic topics from textual reviews, and the target features are captured based on the most relevant words of each discovered topic. Their evaluation shows that topic modeling method can organize information provided by users in subcategories that facilitate the understanding of which features more positively/negatively impact the overall rating of the app.

2.2.3 Assistance in Release Planning

Decisional process for assigning features to subsequent releases under technical, resource, risk, and budget constraints is one of the most critical activities in software product development [62]. The decision-centric process of mobile apps is referred to as app release planning [157]. Mobile app platforms allow developers to deploy and rapidly update their apps. According to McIlroy *et al.*'s study [117], around 15% of the studied 10,713 mobile apps are updated on a bi-weekly basis or more frequently. The authors also obtain that users tend to highly rank frequently-updated apps instead of being annoyed about the high update frequency. Although the authors in [116] show that neither higher numbers of releases nor shorter release intervals correlate strongly with changes in success, the quality of app updating indeed influences an app's destiny, being hot or aborted.

Palomba *et al.* [141] devise a method called CRISTAL for tracking informative crowd reviews onto source code changes. CRISTAL also monitors how many reviews have been addressed and assesses users' reaction (user ratings) to these changes. The study discovers

that developers implementing user reviews are rewarded in terms of ratings. Similarly, Martin *et al.*'s work [116] finds that 33% of the studied app releases cause a statistically significant change in user ratings. Nayebi *et al.* [131] perform surveys with users and developers to understand the common release strategies used for mobile apps. Their study achieves that an app's release strategy is a factor that affects the ongoing success of mobile apps. Thus, providing developers with timely and accurate information for app updating is a critical and beneficial task.

Based on the prior study [139], Guzman *et al.* [75] propose a taxonomy for classifying app reviews into categories relevant for software evolution. However, their method relies on manually-labeled datasets. Villarroel *et al.* [185] automatically prioritize the categorized reviews to be implemented when planning the subsequent app release. Generally, the suddenly-increasing app issues are more important for developers. Identifying version-based bursty app issues is a tough problem due to the lack of real datasets with complete version information and labels. Most current work focuses on observing the trends of app issues over time. For example, Vu *et al.* [186] detect sudden issues by counting the most related keywords. Since a single word may be ambiguous without contexts, their follow-up work [187] 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.

2.2.4 Cross-Platform Study

Various studies have focused on the similarities and differences among different mobile application platforms. Tor-Morten *et al.* [71] utilize a mobile game app to compare the four platforms (*i.e.*, Android, Windows Phone, iOS, and Firefox OS) in terms of technical functionality, APIs, development effort, development support and

deployment to live devices. In [177], Tracy discovers the differences of development environment for iOS and Android OS. Benenson *et al.* [44] conduct an online questionnaire to compare users on different platforms based on the demographic differences, security and privacy awareness. In Luo *et al.*'s work [109], they investigate the security impact of UI-based APIs in the WebView component for Android, iOS, and Windows Phone. Ahmad *et al.* [36] examine the security requirements on Android and iOS platforms with respect to the application sandboxing, memory randomization, encryption, data storage format and built-in antivirus. Liu *et al.* [106] explore the Internet streaming access on Android and iOS by analyzing a server-side workload collected from a top mobile streaming service provider. In Zhou *et al.*'s work [202], they identify the different topics and attributes on different platforms (*i.e.*, desktop, Android, and iOS) from bug reports. Different from the previous work, our study aims at discovering the essential issues from users' perspective to better facilitate the app development on different platforms.

2.3 Ad Cost Exploration

The performance cost of mobile apps is one of the major concerns for both developers and users. Taking the energy cost as an example, [114]'s study finds that practitioners would like oracles that can detect energy issues as they occur, instead of waiting for battery drain to become evident. [143] characterize real-world no-sleep energy bugs and automatically detects these bugs based on dataflow analysis algorithm. To measure the energy consumption of mobile apps, [82] and [137] propose hardware-based and software-based methods, respectively. A comprehensive discussion about energy consumption in Software Engineering can be found in the work by [81]; in the following, we center our discussion on ad cost related work.

Mobile ads can also generate several types of costs for end users,

e.g., battery drainage [128], privacy leakage [53, 152, 120], and traffic data cost [148]. According to the research by [95], privacy and ethics, and hidden cost are the two most negatively perceived complaints (and are mostly in one-star reviews) among all studied complaint types. The work [166] shows that malicious ads can infer sensitive information about users by accessing external storage. [172] investigates the effect on user privacy of popular Android ad providers by reviewing their use of permissions. The authors show that users can be tracked by a network sniffer across ad providers and by an ad provider across applications. The study [73] proposes several lightweight statistical approaches for measuring and predicting ad related energy consumption, without requiring expensive infrastructure or developer effort. [191] and [129] discover that the "free" nature in free apps comes with a noticeable cost, by monitoring the traffic usage and system calls related to mobile ads. The work [179] achieves that although user's information is collected, the subsequent use of such information for ads is still low. [158] also explores how many ad libraries are commonly integrated into apps, and whether the number of ad libraries impacts an app's ratings. The authors find that no evidence that the number of ad libraries in an app is related to its possible rating in the app store, but integrating certain ad libraries can negatively impact an app's rating.

To alleviate these threats, [124] and [182] develop a system for enabling energy-efficient ad delivery. In the work [164], the authors propose an architecture MASTAds for allowing ad networks to obtain only the necessary information to provide targeted advertisements with user privacy preserved. An interesting empirical study given by [74] exhibits obvious hidden costs caused by ads from both developers' perspective (*i.e.*, app release frequencies) and users' perspective (*e.g.*, user ratings). In [159], the authors further validate that ads-supported apps use more resources than their corresponding paid versions with statistically significant differences.

The closest studies to our work are [158], [159] and [74]. Different from [158], we propose method to measure user concerns, instead of using user ratings directly. Also, we analyze the correlations between user concerns and performance cost of ads. Different from [159] and [74] which focus on demonstrating whether ads can bring more hidden costs, we aim at exploring what cost types users actually care about, and how the hidden performance costs of ads can affect user opinions. We conduct large experiments and case studies to answer these questions and provide suggestions for app developers.

□ **End of chapter.**

Chapter 3

Tracking the Changes of User-Concerned App Aspects

Tracking the quantitative changes of app aspects/topics discussed in user reviews can help developers timely understand user demands along with time periods. This chapter presents an aspect-tracing mechanism based on top-ranked reviews. The key notion is that the proposed review-ranking method does not need labeled reviews to filter the non-informative ones out. A case study on the public Facebook Android app indicates that our mechanism can successfully expose one crucial issue of the app. The main points of this chapter are as follows. (1) It presents a review-ranking method based on prioritized topics. (2) It conducts extensive experiments to evaluate the effectiveness of the proposed review-ranking method. (3) It demonstrates the advantage of analyzing user reviews over time.

3.1 Introduction

User reviews are written by app users to express their experience with specific app versions. One piece of user review generally contains the post date, user rating, review text, review title, and app version. The rich information delivered by reviews helps developers monitor the app issues encountered by users timely and further

improve the quality of their apps in the next release. For instance, with review-based app modification, the game app, Flappy Bird, shot to the top of the App Store with zero marketing spent, estimated to cost over $80,000 through customer acquisition.

The reviews are usually massive in size, shorter in length, and contain massive useless information, thus leading to great challenges for developers to identify the key reviews of interest with manual labor. For example, the public Facebook Android app receives around 10,000 reviews per day [2]. Handling the reviews manually is prohibitively time-consuming for developers, especially for popular apps.

Besides, only 35% reviews are informative [52], where "informative" reviews imply the reviews contain information that app developers are looking to identify and is potentially useful for improving the quality or user experience of apps. The effectiveness of removing non-informative reviews in surfacing important topics has been demonstrated in [52]. Although the supervised methods for judging the informativeness of a review save much more time than purely manual inspection (7.4 hours), they still consume around 0.5$\sim$0.9 man-hours on inspecting 2,000 Facebook reviews. How to extract valuable information from user feedback is a critical problem yet to be well addressed. In this chapter, to automate the review filtering process, we propose a review-ranking method based on prioritizing topics, during which reviews conveying important topics are ranked higher.

Moreover, user reviews are continually updating along with rapidly-evolving app versions. They usually reflect user experience of specific app versions. Previous work generally focuses on extracting topics [52, 89, 76] or empirical study [77, 90] based on a static collection of user reviews, ignoring the timeliness of reviews. Tracing the changes of user-concerned app aspects along with time is helpful for developers to design the next release, such as determining the bugs to fix, new functionalities to add, and features

to improve. To fill this significant gap, we trace the quantitative changes of app aspects based on the top-ranked reviews of different time periods.

Specifically, we propose a tool named "AR-Tracker" (App Review Tracker) for prioritizing user-concerned reviews and tracing the changes of app aspects along with time. The review-ranking method is built on topic extraction and topic ranking. We compare five topic modeling algorithms for determining the most effective topic extraction methods. For prioritizing topics, we consider semantic relevance between the extracted topics and also user ratings. A case study on the Facebook Android app shows the usefulness of tracking app aspects over time in timely identifying important app issues. We also visualize the aspect-tracking results in an intuitive and interpretable way, so developers can observe the trends of hot issues clearly. Experiments on four popular mobile apps indicate that our review-ranking method achieves comparable accuracy with the state-of-the-art method [52], while guaranteeing the informativeness of top reviews.

In summary, this chapter makes the following contributions:

- We propose topic-ranking and review-ranking methods for prioritizing important and informative user reviews, which can help developers capture the most user-concerned app aspects.
- We establish a tool, AR-Tracker, to trace and visualize the changes of top-ranked reviews over different time periods, and show a case study on the usefulness of such tracking.
- We evaluate the effectiveness of the proposed review-ranking method on a large-scale experiment involving over 500,000 user reviews of four popular Android apps.

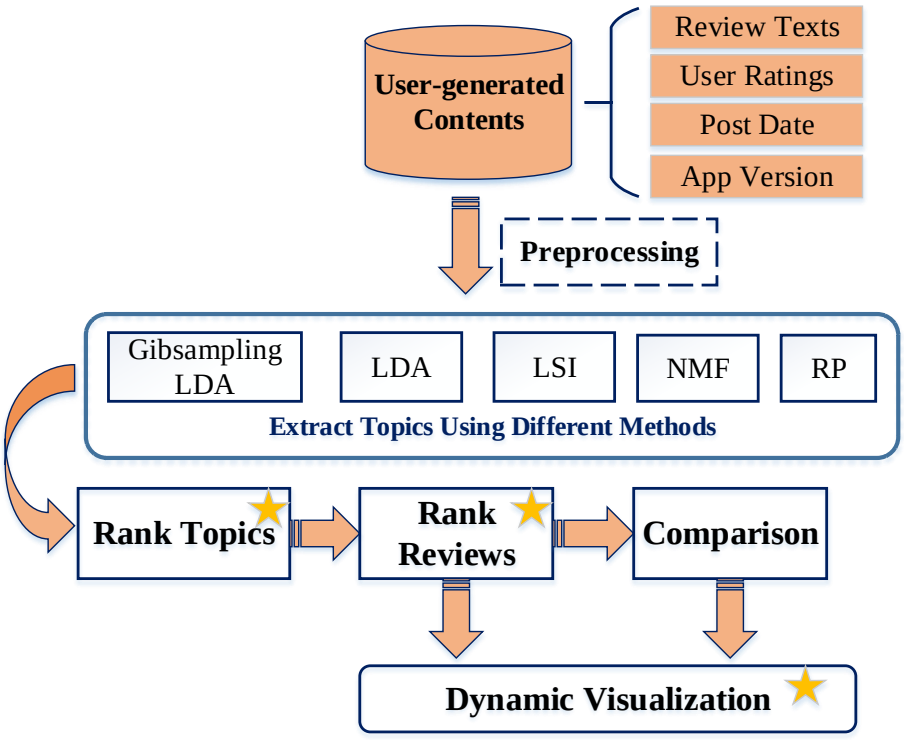


Figure 3.1: Overview of AR-Tracker

3.2 Methodology

This section first gives an overview of the proposed tool, AR-Tracker, and then elaborate each of the five procedures in AR-Tracker, including review preprocessing, topic extraction, topic ranking, and review ranking.

3.2.1 Overview of AR-Tracker

The general framework of AR-Tracker is illustrated in Fig 3.1, which comprises 5 main steps. To begin with, we need to collect user reviews of subject apps from Google Play market, and preprocess the raw user reviews to clean reviews for subsequent analysis. The second step is to extract topics from preprocessed reviews using topic modeling methods. We incorporate five different topic

Table 3.1: Example of User Review Instances

Review ID	Review Text	Rating	Date	Version
1	Never had an issue with it	5.0	11/08/14	21.0.0.23.12
2	Hate that I have to download.	1.0	09/08/14	20.0.0.25.15
3	Can't download videos.	2.0	07/14/14	12.0.0.15.14
...	...	...	...	...
n	r_n	a_n	t_n	v_n

Notes: Each row means a review instance, including a review text, user rating, post date and the corresponding app version.

modeling methods to observe their performance in topic extraction. The topic modeling methods will be briefly introduced in the topic extraction procedure. Then, we illustrate our proposed topic-ranking scheme to prioritize the extracted topics in order of importance. In the next step, on the basis of the ranked topics, we prioritize the user review instances. We employ the official website of the subject app as ground truth for comparing different topic modeling methods in review ranking. Finally, we manually extract user-concerned app aspects from top-ranked reviews, and visualize the qualitative trends of the app aspects along with different time periods.

3.2.2 Problem Setting

Considering an individual app $\mathcal{A}$ from a certain app store, it involves a list of n review instances with review texts $R = \{r_1, r_2, \cdots, r_n\}$, user ratings $A = \{a_1, a_2, \cdots, a_n\}$, post time $T = \{t_1, t_2, \cdots, t_n\}$ and corresponding versions $V = \{v_1, v_2, \cdots, v_n\}$. Therefore, for each review instance r_i, we have its attributes: post date t_i, rating a_i, and version v_i. The general structure of data is designed as $\mathcal{A} : \{R, A, T, V\}$. Table 3.1 shows the notations of all the variables and a review sample with n review instances respectively.

To track the changes of app aspects along with time, we divide app reviews into several time sequences $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_z\}$, where z indicates the number of time periods.

3.2.3 Step 1: User Review Collection and Preprocessing

We choose four subject popular Android apps from Google Play, including the public Facebook (social app), Facebook Messenger (communication app), TempleRun2 (action game app), and Instagram (social app). The reason why we choose these apps are that they belong to different categories. Also, some of them (Facebook, Facebook Messenger, and Instagram) are possessed by the same company, so that we can expect to detect some app issues of the company. Most importantly, they are prevalent worldwide, making sure the quantities of reviews in different time periods are considerable and sufficient for app aspect tracking.

We collected meta-information and user reviews of mobile apps by (a) building a customized web crawler using a web-automation and testing tool `Selenium` based on Python; (b) and utilizing officially-provided app crawling APIs to capture user reviews given apps' package names.

We conduct basic preprocessing methods to clean the raw user reviews, such as taking their lower cases, removing the stop words provided by NLTK [17], and filtering out reviews that contain no words. The preprocessed reviews are utilized for subsequent analysis.

3.2.4 Step 2: Topic Extraction

Extracting topics in a large number of user reviews is helpful to efficiently understand user-concerned app features. Topic modeling methods are typical statistical methods for analyzing the words of the original texts to discover the main delivered topics [45]. They do not require any prior annotation or labeling of the documents - the topics emerge from the analysis of the texts. Various topic modeling methods have been proposed based on probabilistic graph models, such as Latent Semantic Indexing (LSI) [58], Latent Dirichlet Allocation (LDA) [49], Random Projection (RP) [94], and Non-

Negative Matrix Factorization (NMF) [102], and Gibbs Sampling of LDA [196]. Previous research directly utilized the popular topic modeling method - LDA [52, 76] or other feature clustering methods [187, 77], and rarely explores which topic modeling method is better for app review mining. We compare topic extraction from user review instances with different topic modeling methods in this step.

3.2.5 Step 3: Topic Ranking

In order to discover the important topics for developers, we need to prioritize the extracted topics from Step 2. Existing methods merely consider the explicit review attributes, such as user rating, post date, and the number of duplicates, *ect.*, and ignore the implicit relations between these topics. Generally, the topics with larger semantic relevance with other topics would likely to be more concerns of users. Thus, we propose a topic-ranking method by combining the implicit information with explicit user ratings.

Given a list of review texts $R = \{r_1, r_2, \cdots, r_n\}$, we can simply obtain the corresponding vocabulary $D = \{\omega_1, \omega_2, ..., \omega_d\}$ (d is the magnitude of the vocabulary, ω means one specific token). Topics $\beta = \{\beta_1, \beta_2, ..., \beta_k\}$ (k is the number of topics) represent the topics extracted through topic modeling. A review text r can also be expressed as a probability distribution over the topics β, as shown in Table 3.2.

Table 3.2: Review-Topic Matrix

	β_1	β_2	$\cdots$	β_k
r_1	p_{11}	p_{12}	$\cdots$	p_{1k}
r_2	p_{21}	p_{22}	$\cdots$	p_{2k}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
r_n	p_{n1}	p_{n2}	$\cdots$	p_{nk}

The topic-ranking method involves two factors, *i.e.*, one is the semantic similarity with other topics, and the other is the corresponding user ratings. The score S of the i th topic can be

represented as $\mathbf{S}(\beta_i) = \{S(d_i), S(a_i)\}$, where $S(d_i)$ and $S(a_i)$ indicate the scores stemmed from the above two influence factors. Next, we will introduce the detail of the computation.

Topic Similarity

Commonly, ranking and clustering are regarded as orthogonal techniques [173]. Intuitively, if one topic has a larger similarity with other topics, this topic tends to have more significance than the other and can be ranked higher. As shown in Fig. 3.2, Topic C seems closer to the other topics than Topic A or Topic B does, which implies that the words in Topic C are more related to those in B and C. Here, we introduce Hellinger distance, a statistical method to quantify the similarity between two probability distributions.

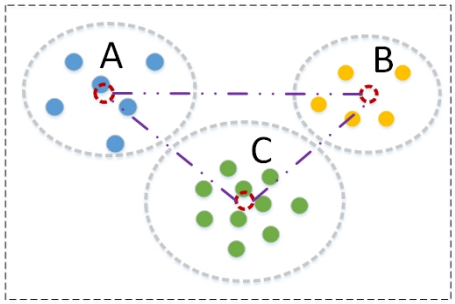


Figure 3.2: Explanation of Topic Similarity. **Note:** The clusters A, B and C indicate three topics. And the dots inside each cluster represent the words belonging to that topic.

For any two topics β_i and β_j, their discrete probability distributions among review instances are $\beta_i = (p_{i1}, \cdots, p_{in})$ and $\beta_j = (p_{j1}, ..., p_{jn})$ respectively, where n means the number of review instances. The Hellinger distance between these two topics is defined as:

$$H(\beta_i, \beta_j) = \frac{1}{\sqrt{2}} \sqrt{\sum_{u=1}^{n} (p_{iu}^{\frac{1}{2}} - p_{ju}{}^{\frac{1}{2}})^2} \tag{3.1}$$

The score of topic similarity for the topic β_Q, $Q \in [1, k]$, is the inverse distance to all the other topics, that is:

$$S(d_Q) = \frac{1}{\sum_{i=1}^{k} H(\beta_Q, \beta_i)} \tag{3.2}$$

where k is the number of the topics.

User Rating

Generally, user reviews with lower ratings imply the users are highly frustrated with some features of the app. And reviews with medium or higher ratings indicate the corresponding features are not very critical.

We define $\mathcal{I}_1$ as the set of reviews with rating 1.0 or 2.0, $\mathcal{I}_2$ as the set with rating 3.0, and $\mathcal{I}_3$ as the set with rating 4.0 or 5.0. Then the score of user ratings for the topic β_Q can be described as:

$$S(a_Q) = w_1 \sum_{i \in \mathcal{I}_1} P_{iQ} + w_2 \sum_{i \in \mathcal{I}_2} P_{iQ} + w_3 \sum_{i \in \mathcal{I}_3} P_{iQ} \tag{3.3}$$

where w_1, w_2 and w_3 are the weights corresponding to each index set $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ respectively and $w_1 + w_2 + w_3 = 1$, $0 \leq w_3 \leq w_2 \leq w_1 \leq 1$. In this way, topics with lower ratings are given higher scores.

Overall Score

In terms of topic similarities and user ratings, the overall score of the topic β_Q is defined as:

$$\mathbf{S}(\beta_Q) = (S(a_Q) + \gamma_1) \cdot (S(d_Q) + \gamma_2), \tag{3.4}$$

where γ_1 and γ_2 are the weights that can be modified according to individual requirements and $0 \leq \gamma_1, \gamma_2 \leq 1$. The parameters are experimentally defined in our situation. We use multiplication for combining the two factors for highlighting their impact on the overall score.

3.2.6 Step 4: Review Ranking

Our goal is to provide developers with more interpretable and direct results, so only a few undefined topics are not satisfying. We need to prioritize the review instances among massive raw user reviews and show the representative ones to developers.

Given a list of review texts $R = \{r_1, r_2, ..., r_n\}$ and their rating information, we get the topic probability distributions of each review text in Table 3.2 and the sorted topics in the last step. The review-ranking method involves three elements, *i.e.*, topic importance score, user rating and semantic similarity to other review instance. We adopt the probability distributions among different reviews to measure the similarity. The grade G of the i-th review instance can be described as $\mathbf{G}(r_i) = \{G(m_i), G(a_i), G(d_i)\}$, where $G(m_i)$, $G(a_i)$ and $G(d_i)$ denote the grades calculated for the above three elements respectively. The detail is described as below.

Topic Importance

For a review text, larger probability distributions in more important topics mean that the instance also tends to be more important for developers. Thus, we group reviews according to the topic probability distributions in a simple mode, which means that one review text belongs to the topic with the largest proportion. Thus, we have k groups of reviews, each with n_j (where $j \in [1, k]$, k is the number of topics) review texts. The score of topic importance for the review text r_Q can be defined as:

$$G(m_Q) = \frac{\sum_{j=1}^{k} \lambda_j \cdot p_{Qj}}{k}, (0 \leq \lambda_j \leq 1), \tag{3.5}$$

where k represents the number of extracted topics. λ_j means the weight to the j th topic and $\lambda_j = \frac{n_j}{n}$, where n is the total number of review instances. p_{Qj} is the probability distribution of j th topic to the review text r_Q.

User Rating

Analogous to the impact of user ratings on prioritizing topics, user ratings also influence the importance order of review instances. Here, we adopt the similar scoring method as the one described in the last step. That is, $\mathcal{I}_1$ means the review group with rating 1.0 or 2.0, $\mathcal{I}_2$ with rating 3.0, and $\mathcal{I}_3$ with rating 4.0 or 5.0. The score of user ratings $G(a_Q)$ for the reveiw r_Q is described as:

$$G(a_Q) = w_i, i \in [1, 3] \tag{3.6}$$

where w_i is the wight to the i th review group. Generally, $w_1 + w_2 + w_3 = 1$, $0 \leq w_3 \leq w_2 \leq w_1 \leq 1$. A larger w_i means the review group with a certain rating has more priority. Equation 3.6 indicates that reviews with lower ratings will be ranked higher.

Review Similarity

If a review text has more resembling texts, the topic reflected by the review would tend to be significant, as the topic is expressed by a majority of users. Thus, this review instance can be ranked higher. A review text can be regarded as a probability distribution over different topics, as shown in Table 3.2. Here, we also adopt the Hellinger distance to measure the similarity between reviews. The score of review similarity $G(d_Q)$ for the review r_Q is defined as:

$$G(d_Q) = \frac{1}{\sum_{i=1}^{n} H(r_Q, r_i)}, \tag{3.7}$$

where n is the number of user reviews in R. $H(r_Q, r_i)$ means the Hellinger distance between r_Q and all the reviews in R. Equation 3.7 indicates that the reviews that are semantically closer to other review texts tend to be ranked higher.

Overall Score

With respect to the above three elements, *i.e.*, topic importance, user rating, and review similarity, the overall score $G(r_Q)$ of the review

r_Q is denoted as:

$$\mathbf{G}_Q = (\alpha_1 \cdot G(m_Q) + \alpha_2 \cdot G(a_Q)) \cdot G(d_Q), \tag{3.8}$$

where $\alpha_i (i \in [1, 2])$ means the weight to each factor and $(0 \leq \alpha_1, \alpha_2 \leq 1)$. Developers can adjust them according to their own demands.

3.3 Evaluation Study

We evaluate our framework AR-Tracker on a large sample of online app reviews crawled from Google Play during August∼November in 2014. Table 3.3 lists the subject apps and the number of reviews in each time sequence of these apps. In the first column of the table, "11-01" means the time period from "11-01" to the latest date of the reviews; "10-15" indicates the period from "10-15" to "11-01". The total number of reviews is 555,529, including 176,362 Facebook reviews, 205,505 Facebook Messenger reviews, 42,807 TempleRun2 reviews, and 130,855 Instagram reviews. We manually divide the reviews into seven time slices for tracking the app aspect changes along with time periods. Our experimental study aims to answer two research questions:

RQ1: Which topic modeling method can better extract the topics from user reviews and whether our review ranking method outperforms the state-of-the-art method AR-Miner?

RQ2: Can AR-Tracker effectively track the quantitative changes of the app aspects reflected in user reviews?

Also, we utilize the dataset of SwiftKey (6,282 reviews in total) used in [52] for comparison of different topic modeling methods and evaluation of AR-Tracker, *i.e.*, answering RQ1. We introduce our performance metrics and answers to the two research questions in the following sections.

Table 3.3: Experimental Dataset

	Facebook	Facebook Messenger	TempleRun2	Instagram
11-01	12,678	14,515	2,531	11,836
10-15	31,787	29,715	7,312	31,797
10-01	26,690	23,966	6,259	26,035
09-15	23,538	30,143	7,195	27,315
09-01	29,096	39,793	6,901	28,144
08-15	31,303	40,510	7,933	5,728
08-01	21,270	26,863	4,676	-

3.3.1 Performance Metric

In this part, we introduce the performance metrics used for our experimental evaluation. To compare with the state-of-the-art method AR-Miner [52], we adopt the same metrics such as Hit-rate (recall) and Normalized Discounted Cumulative Gain NDCG@k [57]. They are calculated as below.

$$Recall(Hit-rate) = \frac{TP}{TP + FN}, \tag{3.9}$$

where TP, FN indicate the numbers of true positives (hits) and false negatives (misses), respectively.

$$NDCG@k = \frac{DCG@k}{IDCG@k}, \tag{3.10}$$

where $NDCG@k \in [0, 1]$, and a higher value implies a stronger agreement between the predicted rank order and the true rank order.

Also, since our framework does not filter the non-informative user reviews, we use self-defined Info-rate as one index for analyzing the proportion of informative reviews in the top reviews.

$$\text{Info-rate} = \frac{\#\text{informative reviews}}{\#\text{top reviews}} \tag{3.11}$$

where $\text{Info-rate} \in [0, 1]$, and the higher value implies more informative reviews are included in the top reviews.

The ground truth for evaluating the proposed review-ranking method is similar to that for evaluating AR-Miner. The ground truth is the *SwiftKey feedback forum* created by the developers. Feedback forum provides users a voting mechanism for every feedback, and feedback with high-voting is ranked top. Since we use the same datasets of Swikftkey as AR-Miner, we also adopt their snapshot from the user forum as ground truth, as shown in Table 3.4.

Table 3.4: Top-10 ranked results attained from SwiftKey feedback forum [52].

Rank	Votes	User Facebook
1	5711	More themes. More themes. More themes
2	4033	Continuous input - glide your fingers across the screen / Flow
3	4025	Option to disable auto-space after punctuation and/or prediction
4	3349	customizable smileys / emoticons
5	2924	AutoText - Word Substitution / Macros (brb = 'be right back')
6	2923	Traditional Chinese
7	2504	An option to use swiftkey predictions everywhere in every app including a web searches
8	2313	Chinese pinyin
9	2095	Thai
10	2014	After Jelly Bean update, Swiftkey keeps returning to android default keyboard after restart or shutdown

3.3.2 Answer to RQ1: Effectiveness of AR-Tracker

For evaluating the performance of the proposed review-ranking method, we experimentally set w_1, w_2, w_3 as 0.85, 0.1, and 0.05, respectively. The whole parameters are illustrated in Table 3.5. Due to the limitation of writing space, we just describe the top five results of AR-Tracker (using Gibbs Sampling LDA during topic extraction), shown in Figure 3.3. We can discover that the fifth review text is non-informative and three of the top five reviews hit the ground truth.

Table 3.5: Experimental parameters.

Parameter	k	α_1, α_2	γ_1, γ_2	w_1	w_2	w_3
Value	10	1	0	0.85	0.1	0.05

Rank	Review	Key Phrase
1	It went off for a repair and came back perfect until I installed swiftkey 3 again.	Update
2	Everything was great, after updating to jellybean stock rom on tmobile I can no longer keep set as my default keyboard, have to set it every time I turn my phone on, getting annoying, please fix.	Jellybean[10]
3	But between it shoving words after commas without me noticing, overly aggressive spell check and if I let it type out a sentence by just hitting space bar a log it types out my cover letter for a job I applied for over a year ago.	Auto-space after punctuation[3]
4	Doesn't predict swears, custom words, or abbreviations w/ symbols (ie. "w/").	AutoText-word substitution[5], Custom words[4]
5	This has happened multiple times which is annoying because I paid for this.	

Figure 3.3: Top five review texts of Gibsampling LDA. The numbers beside the key phrases indicate the phrase rankings in the ground truth in Table 3.4.

To compare with AR-Miner and different topic extraction methods, we use the same datasets of SwiftKey (6,282 user reviews totally) and performance metrics. The results are illustrated in Figure 3.4.

As can be seen from Figure 3.4, Gibbs Sampling LDA can achieve the identical hit-rate as AR-Miner (LDA), and a higher hit-rate than AR-Miner (ASUM). However, pure LDA without the filtering process for removing non-informative reviews does not perform better than the pure GibLDA. In terms of Info-rates and Hit-rates (only applicable to methods without filtering[1]), LSI, LDA, RP, and NMF all present lower performance than Gibbs Sampling LDA,

[1]That is ignoring AR-Miner (LDA) and AR-Miner (ASUM) methods.

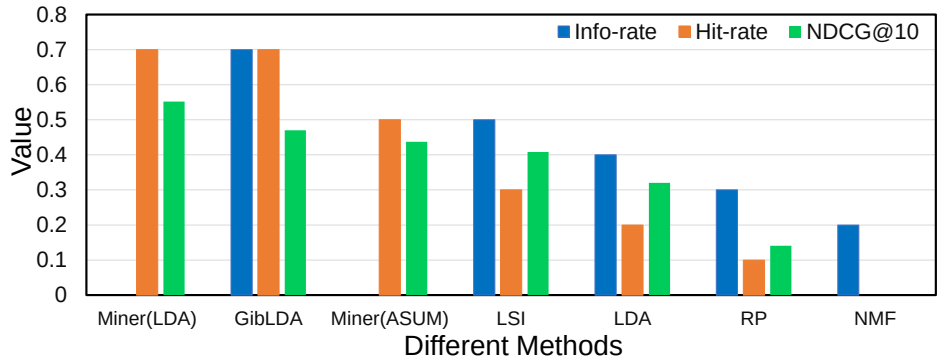


Figure 3.4: Comparison of different methods. We consider the Info-rate for the methods in AR-Miner as 1.0 by default.

which means these methods are more vulnerable to noises (*e.g.*, non-informative reviews). Although the NDCG@10 of Gibbs Sampling LDA is a bit lower than that of AR-Miner (LDA), Gibbs Sampling LDA is higher than AR-Miner (ASUM).

Most importantly, Gibbs Sampling LDA spends developers zero manual labor for labeling informative reviews. In AR-Miner, the total number of reviews amounts to just 6,282 (3,282 for the unlabeled set), far less than the actual number (thousands of reviews per day) for popular apps. As [52] stated, AR-Miner took 0.5 man-hours (for 3,000 labeled data) for EMNB filtering and 7.4 hours for purely manual inspection, while filtering is non-necessary in our framework. Therefore, AR-Tracker is more applicable and practical for the scenario where tremendous numbers of raw reviews need to be analyzed. And we will use AR-Tracker (Gibbs Sampling LDA) for tracking the changes of app aspects from user reviews during answering RQ2.

3.3.3 Answer to RQ2: Tracking changes of app aspects

We analyze the reviews of Facebook over time for illustrating the helpfulness of tracking app aspects. From the top 10 review texts of different time sequences, we extract 10 key topics - Crash,

Newsfeed, Picture, Post, Notification, Privacy, Space, Video, Messenger, and Navigation. We summarize the ranks and frequencies of occurrence of these themes in a specific time period, depicted in Figure 3.5. In the cell of the figure, the number beside the bracket indicates the rank of the topic, and the inside denotes the occurrence frequency of the topic. If the topic only appears once in the top reviews, we just display the rank without bracket.

	08-01	08-15	09-01	09-15	10-01	10-15	11-01
Crash	5(2)	7	8(2)	4	2(3)	4(2)	1(3)
Notification	9	0	0	0	0	0	2
Privacy	6(2)	2(3)	1	0	4	0	0
Space	4	0	0	1(3)	2	5	0
Navigation	8	0	0	2	0	1	4
Newsfeed	8	0	0	3	1(2)	9	3
Picture	0	6	0	5(2)	0	3	4
Post	1	0	0		4	2	8
Video	2	6(2)	0	9(2)	1	10	10
Messenger	1(5)	1(8)	2(5)	1	8(2)	7(2)	0

Figure 3.5: Summary of top 10 reviews in different time periods.

To clearly describe the changes of each topic, we divide the topics into two groups: General issues (with the orange ground in Figure 3.5) and Content issues (with the gray ground), where content issues are specific to app functionalities. And we define $\mathrm{Importance\text{-}rate}$ as below to illustrate the significance level of each topic belonging to content issues.

$$\mathrm{Importance\text{-}rate} = \frac{1 - \mathrm{rank}}{N} * (\lambda + \mathrm{frequency}), \tag{3.12}$$

where rank and $\mathrm{frequency}$ represent the rank and occurrence frequency of the topic in specific time period, respectively. λ is

for regularization and here we experimentally set $\lambda = 0.1$. The Importance-rate of each topic over time is illustrated in Figure 3.6.

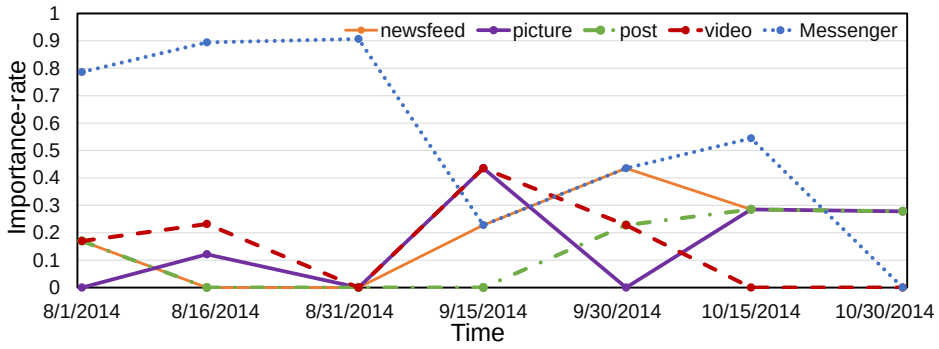
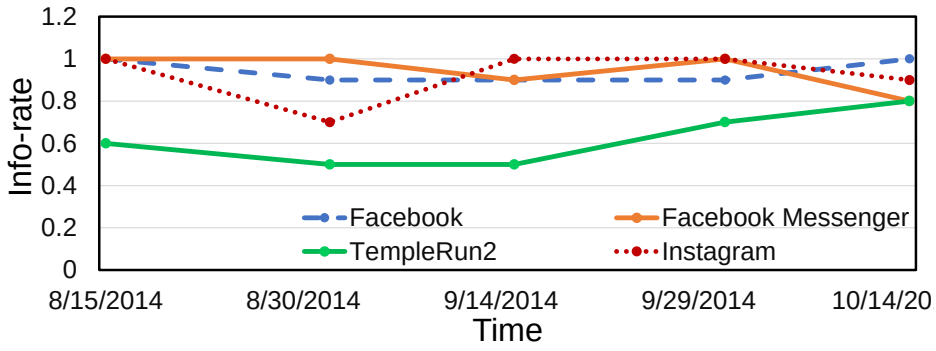


Figure 3.6: Dynamics of Topical Issues

As Figure 3.6 described, the trend of the topic "Messenger" is the sharpest and most interesting one among all the studied topics. This is because users were forced to download Messenger app to check Facebook messages when the Facebook Messenger app first launched, which aroused much discontent and strong response, also reported by [10]. If the developers can identify such topics in a timely manner, they can immediately solve the bugs or improve the features, and thereby the user experience can be guaranteed. Thus, we suppose that tracking the quantitative changes of the top app aspects can really help developers timely detect the most user-concerned issues.

Further, we summarize the average Info-rate of the top-ranked reviews for the subject apps along with time, described in Figure 3.7. It shows that Game app TempleRun2 has the lowest informative rate but not less than 0.5, and all the other apps achieve informative rates higher than 0.8, which indicates the validity of AR-Tracker on avoiding noises without manual labor.

Figure 3.7: Info-rates of Different Apps Over Time



3.4 Summary

User reviews are a favorable and crucial repository for mobile app developers. Since the user reviews are normally massive and messy for popular apps, manual labor is rather time-consuming and inapplicable. In this chapter, we propose a novel review-ranking method without manual labeling for filtering. Also, we produce a new index for measuring the proportion of informative reviews in the top reviews. Furthermore, we visualize the changes of top-ranked app aspects based on these top reviews, and evaluate the tracking results with a case study on Facebook.

There are two major advantages of our framework: (i) it does not need any manual labor and can achieve similar effect as the state-of-the-art method - AR-Miner. (ii) It can track the changes of major app aspects reflected by the top reviews, which would facilitate developers to determine the next commit.

□ **End of chapter.**

Chapter 4

Prioritizing App Issues over Versions from App Reviews

Automatically tracking the app issues, such as laggy user interface, high memory overhead, and privacy leakage, along with app versions is helpful for developers to discover the important ones in current versions. This chapter presents PAID, a framework designed for prioritizing app issues with minimal manual labor and good accuracy. The key notion is that PAID presents app issues in the level of phrase (*i.e.*, a couple of consecutive words), as phrases can be more efficiently understood by developers than long review sentences. The main points of this chapter are as follows. (1) It presents an issue-ranking method systematically and automatically, where the app issues are represented with phrases. (2) An evaluation metric is proposed to measure the accuracy of app issue prioritization by employing the official app changelogs. (3) It evaluates the effectiveness of PAID on a real large-scale dataset.

4.1 Introduction

Different from traditional software resources such as source code and documentation, user reviews on mobile apps are resources directly from customers and can be exploited by developers during the bug-fixing and feature-enhancing process. They are time-sensitive,

that is, the important reviews are varying along with different app versions. For example, according to the official announcement of WhatsApp, a popular communication app, it commits a new version every 3.77 days on average from November 2014 to January 2015, as shown in Figure 4.1. The ratings fluctuate with the version updates, indicating that app versions indeed affect user feedback. Although there exists work [163] focusing on discovering the migration of general app properties (*e.g.*, "Games" shows a tendency to "Spots & Recreation", and "Finance" grows close to "Business"), few studies have explored the trends of concrete app features from user reviews.

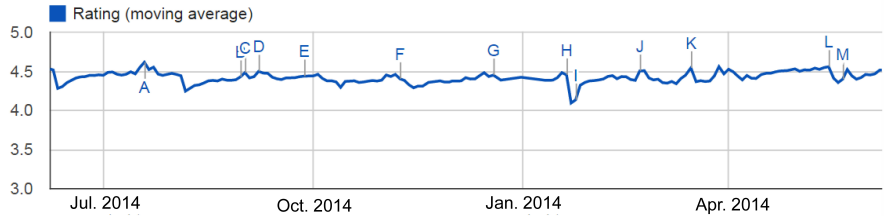


Figure 4.1: Rating changes along with different official releases of Android WhatsApp Messenger app [3]. The capitals above the line indicate the main version releases of the app.

There are several challenges in automating the concrete issue tracking from user reviews. On one hand, filtering meaningless reviews manually or in a supervised manner would be labor-intensive especially to popular apps with tremendous reviews. On the other hand, the topics extracted using topic modeling methods, such as Latent Dirichlet Allocation (LDA) [49], are usually represented as probability distributions over the whole vocabulary. The topic meanings are not intuitive and understandable enough for developers. Thus, automatic interpretation of the topics is helpful and time-saving for developers.

To overcome the challenges, we propose an issue-tracking framework named PAID[1] for prioritizing app issues by tracking user reviews over release versions. Our goal is to facilitate the process

[1] Acronym for Prioritizing App Issues for Developers

of detecting important app issues from app reviews while achieving good performance. Specifically, we propose a phrase-level app issue detection method. We assume that issues represented in phrases can consume developers less time to recognize the issue meanings than in review sentence. Here, one phrase indicates a 2-gram term (*i.e.*, two consecutive words) especially. As the example shown in Figure 4.2, the developers can quickly learn the main aspects of the user's complaints with key phrases (highlighted in dash rectangles) presented. PAID is thereby designed to provide app issues in the level of phrase. Similar to "*Bag-of-Phrases*", we establish a *Phrase Bank*, a dictionary containing all the meaningful phrases in the user reviews.

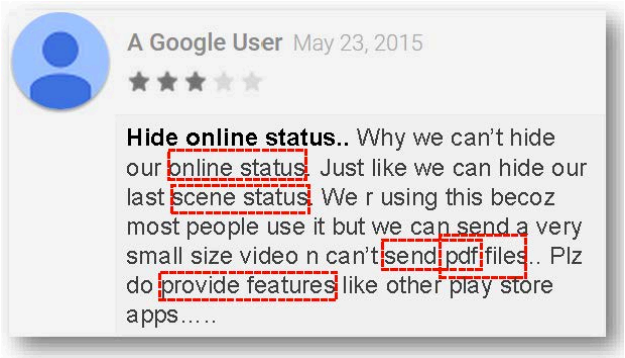


Figure 4.2: An instance of user reviews with useful phrases in dash rectangles.

To automate the exclusion of non-informative reviews, we label 60 useless words, including common emotional words (such as, "like", "amazing", "cool", etc.) and meaningless description words (such as, "facebook", "star", "app", etc.). This labeling process just takes a couple of minutes. Phrases containing the useless words are eliminated. The list of the non-informative words can be easily applied and extended to the preprocessing step of other apps. We employ dynamic topic modeling method [48] to track the topic changes along with different app versions. To interpret the meaning of each topic, we present a topic labeling method to label topic with the most relevant phrase from the *Phrase Bank*. In

this way, developers can directly understand the extracted topics via the phrase labels. For facilitating the observation of quantitative changes of the topics, we also present a visualization way based on ThemeRiver [79]. Large-scale experiments on 18 apps with 117 version releases validate the effectiveness of PAID.

In summary, this chapter makes the following contributions:

- We propose an issue-prioritizing framework for ranking phrase-level issues. We mitigate the impact of non-informative reviews by filtering out predefined useless words. A topic labeling method is also presented to automatically interpret the semantic meaning of each topic.
- An evaluation metric based on the official app changelogs is introduced for measuring the performance of app issue prioritization.
- Extensive experiments on large real-world datasets are conducted to validate the effectiveness of PAID in prioritizing user-concerned app issues along with versions.

4.2 Methodology

This section first gives an overview of the proposed issue-prioritizing framework, PAID, and then elaborate on the main three procedures in PAID, which includes data extraction, app issue generation, and visualization and issue analysis.

4.2.1 Overview of PAID

The overall framework of PAID is shown in Figure 4.3, including three main steps. During the data extraction step, we preprocess the raw user reviews crawled from Google Play Store, and filter out the noise words and non-informative reviews. Then, we import the

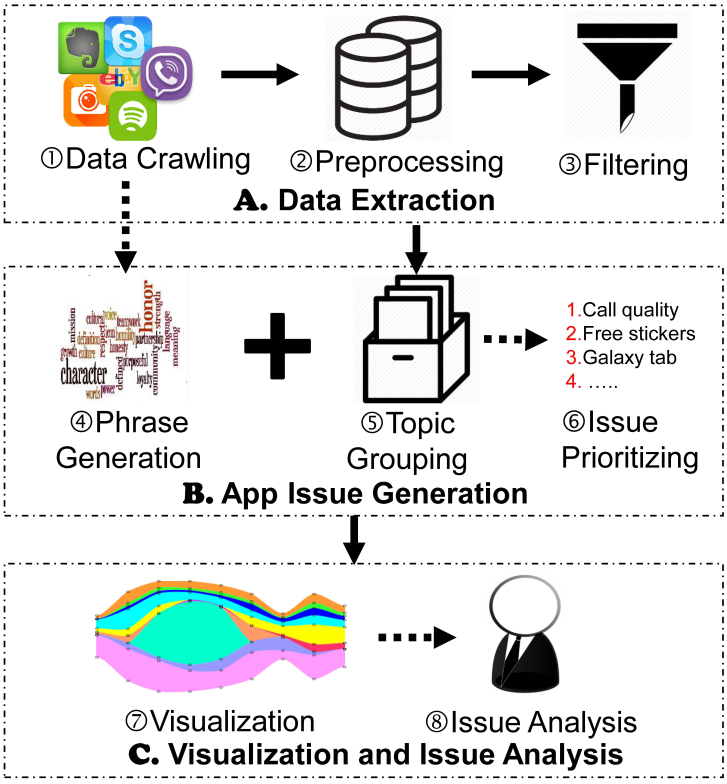


Figure 4.3: The framework of PAID

filtered reviews into the issue-generation segment. In the second step, we first construct *Phrase Bank* and group time-sensitive topics, then we label each topic with the most relevant phrase in *Phrase Bank*. The topic labels are the prioritized app issues. In the last step, we compute the occurrence of each app issue along with versions, and display them to developers through visualization. For developers who want to gain an in-depth understanding of an issue, we also recommend important user reviews correspondingly.

4.2.2 Step 1: Data Extraction

Data extraction serves to format the raw data and remove the useless information for the subsequent analysis. It includes data crawling, data preprocessing, and filtering.

Table 4.1: A Snapshot of the Database

No.	Author	Review	Title	Date	Star	Version
1	Marvin	Not working in Lenovo A606 plus. Fix it.	Help.	2015-05-26T11:00:42	1	5.4.0.3239
2	Sibiya	I love it.	Best app ever.	2015-05-26T10:52:19	5	5.4.0.3239
3	Moham	Nice apps.	Khalil	2015-05-26T10:52:33	3	5.4.0.65524
4	Hassan	Superrrrrrr.	Alcshita	2015-05-26T10:41:01	5	5.4.0.45564
5	Andrew	Would's worst app.	Can't sing till	2015-05-26T10:34:56	1	5.4.0.3239

Data Crawling

PAID employs specific crawling APIs provided by AppFigures [4]. AppFigures supports access to app store repositories, such as reviews, products, user ratings, and app ranks, etc., along with a variety of query filters. It takes us a few weeks to collect the review repositories of the subject apps. Table 4.1 presents a snapshot of the collected dataset. The dataset contains seven attributes for each review instance and we use five of them in this chapter: Number, Review, Date, Star, and Version. The words highlighted in yellow are considered as *non-informative*, as they do not deliver any information potentially useful for improving the quality or user experience of the app. In total, we have crawled 2,089,737 user reviews of 37 apps during a period of 10 months (see details in Table 4.6).

Preprocessing

The preprocessing part and filtering part prepare the dataset for the subsequent topic grouping process, as shown in Figure 4.3. Our goal is to track user reviews over versions; therefore, we need to divide user reviews into different app versions. As some versions possess insufficient reviews for analysis, we combine the consecutive ones

to form a larger review collection.

First, we take the lowercase of all the words in the review texts. Then we reduce the words to the root form by lemmatization [15]. The reason why we do not choose stemming [26] is that stemming crudely chops off the ends of words, which is not suitable for reviews with large numbers of casual words. Meanwhile, lemmatization can preserve the informative derivational ends with the inflectional ends removed. Table 4.2 illustrates this fact (*e.g.*, "occasions" is reduced to "occas" in Stemmer and to "occasion" in Lemmatizer).

First-Layer Filtering

To remove non-English words, we use the typical wordset - Synsets [18] as the first-layer filter. Then we guarantee that the remaining words exclude the stop words in the NLTK [17] corpus.

Table 4.2: Comparison between Stemmer and Lemmatizer

Original Word	Stemmer	Lemmatizer
another	anoth	another
attentions	attent	attention
available	avail	available
compatible	compat	compatible
concentrations	concentr	concentration
occasions	occas	occasion
notifications	notif	notification
solutions	solut	solution

Second-Layer Filtering

As we can observe from Table 4.1, reviews contain casual (*e.g.*, "superrrrrrr") and emotional words (*e.g.*, "nice", "love", and "worst"). To weeding out the non-informative words, we will conduct second-layer filtering. The second-layer filtering part targets at removing non-informative reviews (*e.g.*, "Nice apps", "I love it", etc., as shown in Table 4.1).

In PAID, we simply label 60 meaningless words that frequently appear in the non-informative reviews, such as emotional words (*e.g.*, "annoying" and "awesome"), everyday words (*e.g.*, "good" and "much"), etc. We group these words into *Filter Bank*. The *Filter Bank* used in PAID is listed in Table 4.3. The words in the *bank* are manually identified. Since the number is rather small, it just takes us a couple of minutes to label them. In contrast, it takes about half an hour for the approach proposed in [52] to label hundreds of non-informative reviews. The output of the second-layer filter is fed into the topic grouping process in step 2.

Table 4.3: *Filter Bank* to Filtering Non-Informative Reviews

app, good, excellent, awesome, please, they, very, too, like, love, nice, yeah, amazing, lovely, perfect, much, bad, best, yup, suck, super, thank, great, really, omg, gud, yes, cool, fine, hello, god, alright, poor, plz, pls, google, facebook, three, ones, one, two, five, four, old, new, asap, version, times, update, star, first, rid, bit, annoying, beautiful, dear, master, evernote, per, line.

4.2.3 Step 2: App Issue Generation

The step of app issue generation aims at recommending the most important app issues to developers. We first propose a rule-based method to establish the *Phrase Bank*, which includes all the meaningful phrase candidates. We then cluster the words of the filtered reviews from the last step along with release versions. The app issues are the phrases extracted from *Phrase Bank* for representing the meanings of the word clusters.

Phrase Generation

The foundation of prioritizing phrase-level app issues is to build a *Phrase Bank* (*i.e.*, a phrase collection). Since the preprocessed review texts have been lemmatized and filtered, the meaning of

the phrases generated from these texts may be confusing (*e.g.*, "applic unstabl", "get unlik", etc.). Hence we extract the phrases (specifically referring to 2-gram terms) from the raw user reviews directly instead of the preprocessed reviews by using the rule-based method. Four rules are adopted during this process. First, we use TMI (True Mutual Information) [126] to rank the 2-gram phrases. TMI is defined as the weighted average of the pointwise mutual information for all the observed and expected value pairs, indicating the co-occurrence rate of the words in each pair. Intuitively, a meaningful phrase should frequently occur in the collection, in which the words tend to be highly correlated to each other. The TMI between two words w_1 and w_2 is defined as

$$\mathrm{TMI}(w_1, w_2) = \frac{Pr(w_1, w_2)}{Pr(w_1)Pr(w_2)}, \tag{4.1}$$

where $Pr(w_1, w_2)$ and $Pr(w)$ denote the probability of phrase $w_1 w_2$ and ungram w respectively, and are estimated by their frequencies in the review collections.

Rule 1 (Length Limit) *The length of each word in the phrase must be no less than three.*

Rule 2 (Informative Assurance) *Each word in the phrase should not appear on the stop-word list of NLTK or in the Filter Bank.*

Rule 3 (Part of Speech Limit) *Each word in the phrase should not be an adverb or a determiner.*

Rule 4 (Quality Assurance) *We set a threshold to the probability of $Pr(w_1, w_2)$. The co-occurrence frequency of w_1 and w_2 must exceed five times. Furthermore, we only consider the top 150 phrases based on TMI score. These are to ensure the quality of the phrases in the Phrase Bank.*

Based on the above rules, we obtain the *Phrase Bank*, from which we will recommend the important ones to developers.

Topic Grouping

We prioritize phrases in the *Phrase Bank* to the developers by a grouping-based ranking strategy. First, we adopt a topic modeling method to summarize the words into different topics. Then for each group, we pick a phrase to represent the topic meaning, which can be regarded as a topic *interpretation* process.

To capture the timeliness of user reviews, we use dLDA (Dynamic Latent Dirichlet Allocation) [48] to discover the latent topic sets for each app version. In dLDA, the reviews in the collection are categorized into discrete topics according to their corresponding versions, and the topics evolve as time goes by. A direct explanation of dLDA model is depicted in Figure 4.4, in which the number of app versions is m. For each version, an LDA model [49] is established on the user reviews. The connections between topics of different versions stem from the assumptions of two Dirichlet distributions α and β. Thus we can view the changes of users' attention among versions.

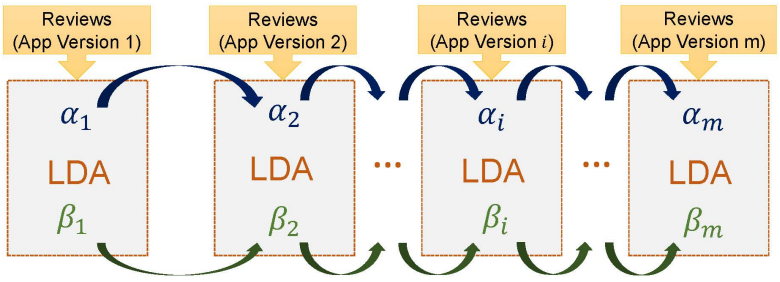


Figure 4.4: A direct explanation of the dLDA model.

Similar to LDA, the number of topics in dLDA also demands to be defined in advance. Preprocessed user reviews in the last step are used as the input of topic modeling. If we set the number of topics of topic modeling to be five, one sample output of user reviews on the Facebook app is shown in Table 4.4. We can observe the transformation of latent topics across different app versions. For example, the top word "video" in Topic 1 disappears during the next

several versions, while the other words such as "fix" and "close" gradually surface.

Table 4.4: One Sample Output of dLDA on Reviews of Facebook

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
Version 1	**video**	use	download	post	updat
	fix	phone	messag	see	work
	time	need	messeng	feed	new
	play	access	take	recent	friend
	close	make	space	news	slow
Version 2	**video**	use	messeng	post	updat
	fix	phone	download	feed	work
	time	need	messag	news	new
	close	access	take	see	load
	keep	permiss	space	recent	slow
Version 3	**fix**	use	messeng	post	updat
	close	phone	download	feed	work
	time	need	messag	news	load
	keep	access	space	see	new
	open	make	take	recent	slow

Issue Prioritizing

From the generated *Phrase Bank*, we select the most representative one for indicating the meaning of each topic produced by dLDA. We recommend the selected phrase-level issues to developers.

The result of dLDA for a specific app version can be displayed in Table 4.5. Given the collection of user reviews $D = d_1, d_2, \ldots, r_n$ (n is the number of reviews), we denote the corresponding vocabulary $W = w_1, w_2, \ldots, w_g$ (g is the magnitude of the vocabulary). After topic grouping, we obtain the probability distribution $Pr(w|\beta)$ for each word w over the topics β. We design a topic-interpretation method from both semantic aspect and sentiment aspect.

Semantic Aspect: A representative phrase for one topic should cover the semantic meaning of the whole topic as much as possible and discriminate across the topics simultaneously. Similar to Mei *et*

Table 4.5: Review-Topic Matrix for a Specific App Version

	β_1	β_2	$\cdots$	β_k
w_1	$Pr(w_1\|\beta_1)$	$Pr(w_1\|\beta_2)$	$\cdots$	$Pr(w_1\|\beta_k)$
w_2	$Pr(w_2\|\beta_1)$	$Pr(w_2\|\beta_2)$	$\cdots$	$Pr(w_2\|\beta_k)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
w_g	$Pr(w_g\|\beta_1)$	$Pr(w_g\|\beta_2)$	$\cdots$	$Pr(w_g\|\beta_k)$

al.'s work [119], we use KL-Divergence to measure the similarity $Sim(\beta, l)$ between the topic β and the phrase candidate l.

$$\begin{aligned} Sim(\beta, l) &= -KL(Pr(w|\beta)||Pr(w|l)) \\ &\approx \sum_{w} Pr(w|\beta) log(\frac{Pr(w, l|c)}{Pr(w|C)Pr(l|C)}), \end{aligned} \tag{4.2}$$

where $Sim(\beta, l)$ is the similarity function, $Pr(w|\beta)$ and $Pr(w|l)$ are the probability distributions over word w respectively generated by β and l, and C indicates the whole review collection. We use the occurrence frequency to calculate the probability $Pr(x|C)$ (x denotes l or w).

$$Pr(x|C) = \frac{\{d \in C | x\, occurs\, in\, d\}}{\{d \in C\}}, \tag{4.3}$$

where d represents one user review. The lower the KL-Divergence is, the more the phrase is semantically similar to the topic. Moreover, to make the phrases across topics to be diverse, we add a term to punish those which present a higher similarity score to multiple topics. Thus, the overall similarity function is modified as

$$Sem(\beta_i, l) = Sim(\beta_i, l) - \frac{\mu}{k-1} \sum_{j \neq i} Sim(\beta_j, l), \tag{4.4}$$

where β_i stands for the current topic to be scored, and μ is used to adjust the penalty due to the similarity to other topics, which is a parameter to be empirically set.

Sentiment Aspect: Intuitively, the developers prefer the reviews with lower ratings and longer lengths. Figure 4.5 reflects this fact. Obviously, the first review provides the developers more information about the app bugs and features, such as enabling the comment functionality and playing video in higher resolution automatically.

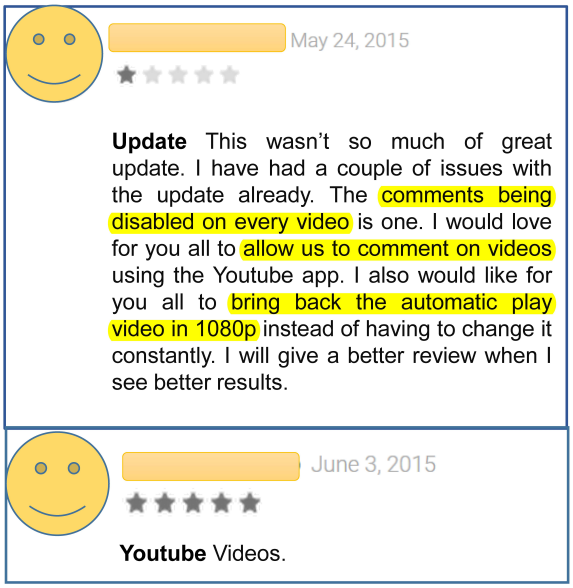


Figure 4.5: Two review instances showing that longer-length and lower-rating reviews are preferred by developers.

For the phrases $l_1, l_2, \ldots, l_z$ (z denotes the number of the phrase candidates) in the *Phrase Bank*, we define their sentiment scores based on user ratings r and review lengths h.

$$Sen(l) = \mathrm{e}^{\frac{-r}{ln(h)}}, \tag{4.5}$$

where r and h of one phrase are defined as the average rating and average length of all the reviews including the phrase, respectively. The phrases contained in the reviews with longer lengths and lower ratings will be scored higher.

Total Score: Combining the semantic aspect with the sentiment aspect by multiplication, the impacts of both factors on the final score are interrelated. We select the phrase l with the highest score

$S(l)$ to represent the topic β_i. The phrases are the app issues we will recommend to the developers for bug fixing or feature improving.

$$S(\beta_i, l) = Sem(\beta_i, l) * Sen(l). \tag{4.6}$$

4.2.4 Step 3: Visualization and Issue Analysis

To help developers better understand the changes of the important app issues over versions, we also involve visualization in the end and provide the issue analysis.

Visualization

We adopt ThemeRiver [79] to represent the transformation of app issues. The technique has been used in handling large document collections [91, 78], but never in the user reviews. The reason we use ThemeRiver is that it provides users with a macro-view of issue changes in the corpus over a serial dimension.

Figure 4.6 shows an example ThemeRiver visualization on Facebook. The "river" flows from left to right through versions, changing its width to depict quantitative changes in the thematic importance of temporally associated reviews. The *width* is denoted by the dash line in Figure 4.6. Colored "current" flowing within the river narrow or widen to indicate decreases or increases in the importance of an individual issue or a group of issues in the related reviews. Here, the importance of the issue is defined as the occurrence frequency of the selected phrase for that issue in the corresponding review collection.

Issue Analysis

We also prioritize user reviews for the developers in the case that they want to gain a deep insight into the phrase-level issue. Similar to the sentiment score of the phrase, the importance score $I(d)$ of the review d is computed based on its length h and user rating r as well.

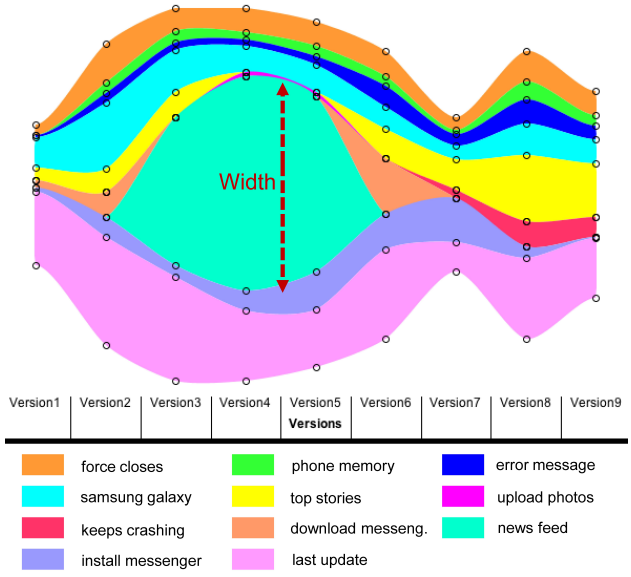


Figure 4.6: A sample ThemeRiver visualization on Facebook. The colored "current" within the "river" indicates an individual issue. The width of the "current" changing with versions denotes the corresponding topic has different degrees of importance for different versions. The issue is represented in phrase.

The number of top reviews to be displayed is determined according to developers' requirements.

$$I(d) = \mathrm{e}^{\frac{-r}{ln(h)}}. \tag{4.7}$$

Finally, the reviews with longer lengths and lower ratings will be ranked higher. Developers can comprehend more about the app issues by checking the reviews in the top list.

4.3 Evaluation

For experimental evaluation, we have crawled more than two million user reviews of 37 apps beginning from August 2014 to June 2015. The subject apps belong to 10 different categories. Each app receives roughly 56,479 reviews on average. With multiple categories and massive reviews, we can verify the effectiveness of PAID while mitigating the data bias from only one type of reviews.

Table 4.6: The Review Dataset of 37 Apps

Category	App Name	Review Quantity
Social	Facebook	176,362
	Twitter	132,981
	Instagram	130,855
Books & Reference	Amazon Kindle	575
	Wikipedia	541
Shopping	eBay	91,368
	Amazon Shopping	792
Photography	Photo Grid - Collage Maker	91,425
	Camera360 Ultimate	79,640
	PicsArt Photo Studio	569
	Autodesk Pixlr - photo editor	500
Tools	Clean Master (Boost & AppLock)	234,342
	Battery Doctor (Battery Saver)	116,534
	CM Security Antivirus AppLock	87,785
Travel & Local	Booking.com Hotel Reservations	29,632
	Google Earth	18,919
	Expedia Hotels, Flights & Cars	1,367
	Foursquare - Best City Guide	494
Communication	WhatsApp Messenger	130,761
	Skype - free IM & video calls	103,479
	Messenger	95,070
	Viber	87,647
	LINE: Free Calls & Messages	70,408
	Chrome Browser - Google	54,707
	Wechat	46,330
	Hangouts	27,515
	Gmail	21,138
	CM Browser - Fast & Secure	500
	Firefox Browser for Android	500
	Contacts+	499
Education	Coursera	608
	Duolingo: Learn Languages Free	487
Productivity	Evernote	48,525
	SwiftKey Keyboard + Emoji	48,028
	ES File Explorer File Manager	507
Music & Audio	YouTude	86,395
	Spotify Music	71,952

Specifically, we aim to answer the following three research questions:
RQ1: What are the trends of app issues reflected by user reviews along with different versions?
RQ2: Would developers modify the issues prioritized by PAID?
RQ3: What is the influences of parameter settings on the performance of PAID?

4.3.1 Answer to RQ1: Case Demonstration

Here, we show the effectiveness of PAID on apps from different categories: Viber (Communication), Evernote (Productivity), Camera360 Ultimate (Photography), and Spotify Music (Music & Audio). The numbers of user comments belonging to various versions of these four apps are shown in Figure 4.7.

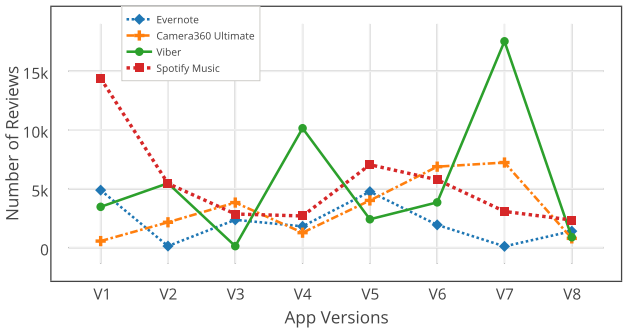


Figure 4.7: Distribution of user reviews for different app versions. We overlook the versions with less than 100 user reviews, and display the latest eight versions in the collection.

Table 4.7 illustrates the top 25 phrases in the *Phrase Bank* of Viber, which is established based on the four proposed rules. In the top list, phrases "video call", "video calls", and "video calling" deliver exactly the same meaning. This is because Viber is characterized by video calling, but developers may not desire such repetitive information. The prioritized phrase-level issues should distinguish from the others in semantics.

Table 4.7: *Phrase Bank* (top 25) of the Viber app

video call, video calls, activation code, samsung galaxy, call quality, phone number, internet connection, free calls, video calling, profile picture, sound quality, public chat, sony xperia, voice quality, online status, globus mobile, asus zenfone, start earning, voice call

Based on the proposed issue-prioritizing method, the phrases possessing the highest total scores are elected for the topics generated by dLDA. We present them with the corresponding scores and along with versions in Table 4.8. The phrases can cover more kinds of app issues (*e.g.*, the phrases for topics of Viber 5.2.2.478 are "free calls", "sony xperia", "animated stickers", "chat background", "activation code", and so on.), and the semantics of the phrases are consistent within one topic (*e.g.*, the phrases for topic 3 are "animated stickers" for Viber 5.2.2.478, "download stickers" for Viber 5.3.0.2274, and "download stickers" for Viber 5.3.0.2331). They are the important issues to be reported to developers ultimately.

Table 4.8: Phrases prioritized for Viber developers ($k = 8, \mu = 1$)

Topics	5.2.2.478	5.3.0.2274	5.3.0.2331
Topic 1	free calls:0.67	free calls:0.66	free calls:0.65
Topic 2	sony xperia:0.97	sony xperia:0.93	sony xperia:0.91
Topic 3	animated stickers:0.53	download stickers:0.56	download stickers:0.63
Topic 4	chat background:0.69	incoming messages:0.70	chat background:0.73
Topic 5	activation code:3.66	activation code:3.69	activation code:3.72
Topic 6	galaxy tab:0.63	galaxy tab:0.62	samsung galaxy:0.61
Topic 7	voice call:1.95	call quality:1.94	call quality:1.94
Topic 8	start earning:1.42	start earning:1.44	start earning:1.45

However, by analyzing so many phrases and figures directly, it is extremely tedious to decide which app issues to modify. Therefore, we apply ThemeRiver [79] to visualize the importance of these app issues. Figure 4.8 displays the ThemeRiver of Viber. The width of the "current" represents the importance score of the issue by counting the occurrence frequency of the corresponding phrase. The issues are in random order within one version, but their positions are consistent over versions. For example, Figure 4.8 demonstrates

that there exists an increasing trend of the issue "activation code", colored in pure blue, for Viber 5.2.1.36, 5.2.2.478, and 5.3.0.2339.

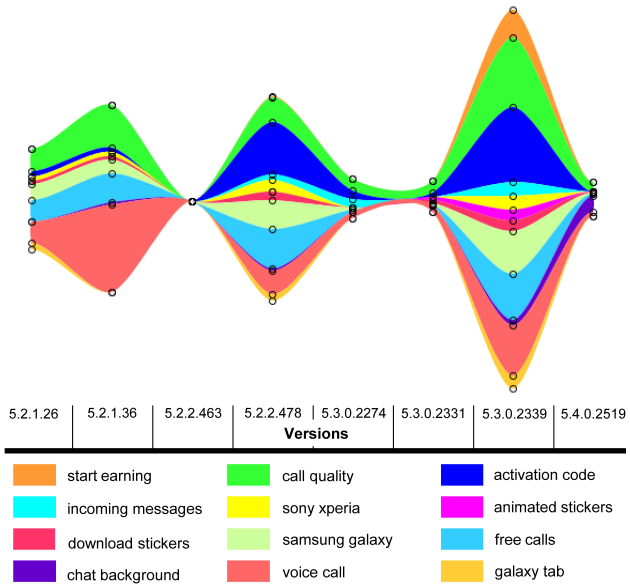


Figure 4.8: Themeriver visualized for the Viber app. The horizontal axis denotes app versions, while the vertical axis means the start point of the "river". Each "current" represents a phrase-level issue. Wider currents stand for more important issues.

To obtain an in-depth knowledge about one issue, such as "activation code", we provide the top reviews associated with the issue based on the method in Chapter 4.2.4. Table 4.9 lists the top three reviews related to the issue "activation code". All these reviews express some bugs of the app and illustrate different aspects (*e.g.*, "Messages are not present" in review 1, "a white popup written only" in review 2, and "keeps on saying activation code sent to your device" in review 3). Therefore, developers can examine the urgent concerns from users by viewing issues first, then decide and analyze them deeply, and finally scheduling the modification.

Table 4.9: Top three reviews related to the issue "Activation Code".

	User Review	Importance Score
1	Upload viber! I went. Enter a phone number. I enter. Asks for sure your phone? It will be sent an activation code. Ok. Messages are not present. He writes to activate viber here, install it to your phone first. But I have it pumped? What to do? Help!	0.836
2	I hard reset my tab 3. Installed viber for activation code when i write my phone number and press okay a white popup written only. ERROR no description given and an okay button on it please help me vibers my only way to contact my son abroad.	0.834
3	I don't know what's wrong with Viber. Just downloaded it nd it keeps on saying activation code sent to your device. For almost a month, no any activation code and it's really pissing me off. Pls fix.	0.828
...	...	...

4.3.2 Answer to RQ2: Performance Evaluation

Case Evaluation

To establish the connection between the analysis of user reviews and the decisions made by developers, we employ the official changelogs of mobile apps. Changelog [6] is a record of all the changes made to a software project, usually including such records as fixed bugs and newly-added features. It is a first-hand and practical ground truth *labeled* by developers directly.

We collect the official changlogs of six versions of Viber from APK4Fun [1] for evaluation. The version of the changelog we compare is the one immediately following the experimental version. For example, to assess the result of Viber 5.2.1.36, we need to inspect the changelog of the next version, *i.e.*, 5.2.2.478. Since our issues are in phrase level, we manually summarize the changelogs into phrases (not limited to 2-gram terms) as Table 4.10. We remove the meaningless phrases and sentences such as "bug fixes", "General stability and bug fixes to improve overall performance.", etc. The highlighted phrases comprise the ground truth of the Viber app.

Table 4.10: Changelogs of Viber and its identified phrases. The phrases highlighted constitute the ground truth, and the strike-through phrases or sentences are discarded.

Versions	Detailed Changelog & Identified Phrases
5.2.1.36	1. Improved sticker menu; 2. Redesigned forward screen gives you the option to send to groups, contacts; 3. Public chats; 4. ~~General stability and bug fixes to improve overall performance.~~
5.2.2.478	1. ~~Bug fixes~~ and updated emoticons.
5.3.0.2274	1. Become an Admin and manage your group chats; 2. Send and receive messages from your watch; 3. Clearer contact info; 4. Public Chat ~~enhancements~~.
5.3.0.2331	Same as the previous version.
5.3.0.2339	Same as the previous version.
5.4.0.2519	1. Enhancements for tablet users; 2. Easier to activate Viber account on your tablet; 3. Improved call and video screens; 4. Send multiple photos more easily; 5. Personalise your groups with your own icon; 6. Customise Viber notifications in Priority Mode on Android L only.

To measure the similarities between the prioritized issues L and the ground truth U, we adopt the sentence-based semantic similarity measure method proposed by Li *et al.* [101]. The method focuses directly on computing the similarity between short texts of sentence length, which fits our situation. We denote the similarity degree between two phrases as $s(u, l)$, where u indicates the phrase in the ground truth, and l means the phrase in the prioritized issues. For each phrase u in the ground truth, we compute its similarity degrees to all the phrases in our results, and the highest one is defined as the rate of the phrase $Rate(u)$, defined in Equation 4.8. The precision of our method is indicated by the average rate of all the phrases in the ground truth.

$$\arg\max_{u} Rate(u) = \{s(u, l) | \forall l : \ l \in L\}. \tag{4.8}$$

Similarly, the precision of the other three apps (Evernote, Camera360 Ultimate, and Spotify Music) can also be calculated. Fig-

ure 4.9 depicts the results of these four apps, with the average precision and standard deviations shown in Table 4.11. All the four apps have precision larger than 55% and two of them (Viber and Camera360 Ultimate) are larger than 70%. Three of the four apps produce the standard deviations less than 0.045, while only the output of Camera360 Ultimate is larger than 0.1. From the Figure 4.9, we can observe that V2 of Camera360 reaches the lowest of its record. By checking the corresponding changelog, we find it is just one sentence "New HDR 'Storm' effect will blow your mind", which is manually identified as "HDR Storm effect" (a kind of technique used in imaging and photography). Contrasting with our prioritized issues ("selfie camera", "tilt shift", "stock camera", "save edited", etc.), we consider the main reason for the result is that the phrase in the changelog is brand-new and has not been embodied in the semantic corpus. However, the acceptable performance of other apps displays the effectiveness of our method. Without the loss of generality, we provide the evaluation of the available apps in our collection in the following chapter.

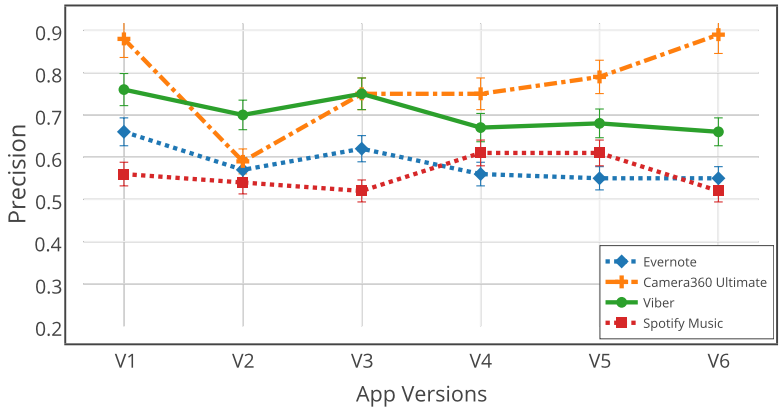


Figure 4.9: Precision of PAID for the apps from four different categories. The experimental parameters for similarity measure settings are $\alpha = 0.002, \beta = 0.3, \eta = 0.6, \phi = 0.8$, and $\delta = 0.9$.

Table 4.11: Average Precision of Four Apps and Their Standard Deviations

	Viber	Evernote	Camera360 Ultimate	Spotify Music
Average Precision	0.703	0.585	0.775	0.560
Standard Deviation	0.042	0.045	0.109	0.041

Generality Evaluation

We employ the apps in Table 4.6 to demonstrate the generality of PAID. Searching the changelogs online, we discover that 19 of the 37 apps do not provide detailed information about version modification. So we adopt the remaining 18 apps with 117 version changes for the verification. The result is shown in Figure 4.10. The average precision for these apps is 60.3%, which is quite acceptable for app review analysis [52].

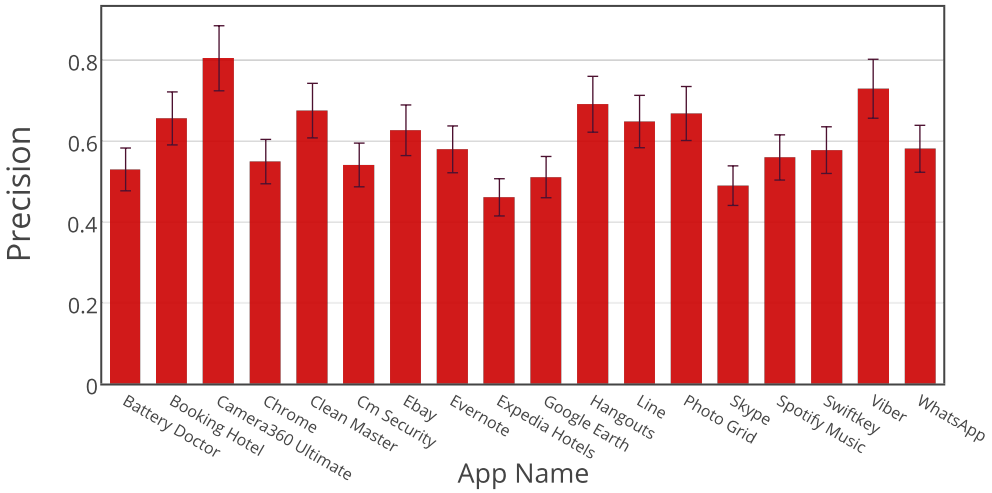


Figure 4.10: Precision of 18 subject apps.

To analyze the correlation between the app category and the performance of PAID, we compute the average precision of the apps belonging to one category and the corresponding average review number (Figure 4.11). We discover that a larger review quantity tends to produce a better result.

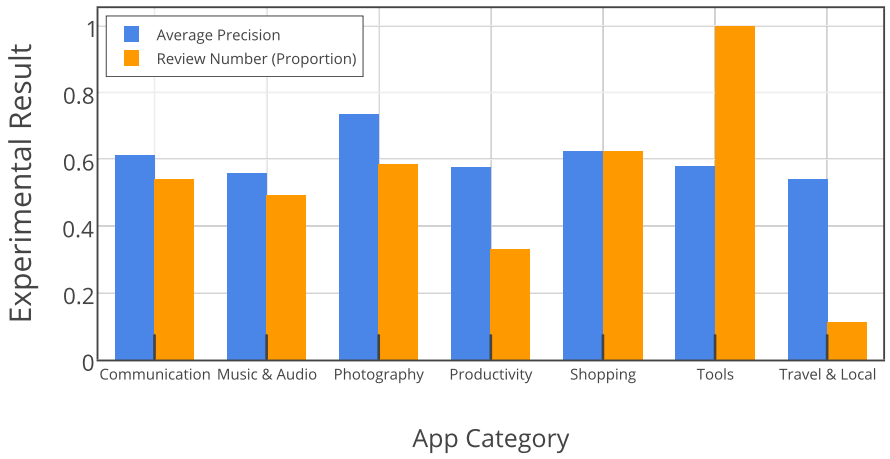


Figure 4.11: Relation between the average precision and the review number regarding to app category. The review numbers have been normalized by the maximum value.

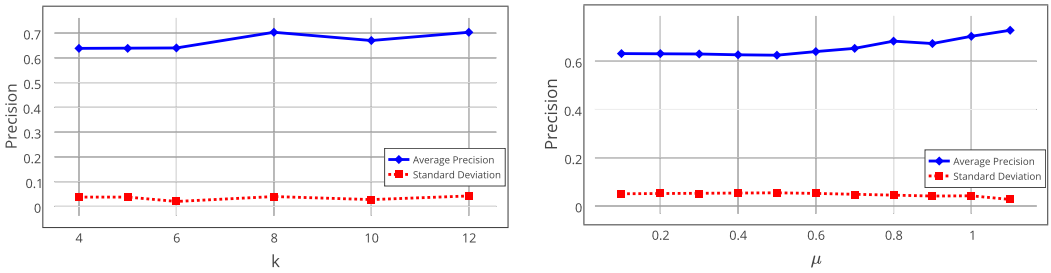
4.3.3 Answer to RQ3: Parameter Study

In this part, we study the influence of parameter settings (*i.e.*, the number of topics k for each version and the penalty factor μ in) for PAID. Figure 4.12 (a) shows that a larger number of topics can produce better precision. This is because more topics generally cover more app issues, and hence possibly contain the issues described in changelogs. Figure 4.12 (b) indicates that more penalty on the dissimilarity to other topics can generate more promising precision and lower standard deviation. Larger penalty makes the prioritized issues more diverse. Thus, more phrases in the changelogs can be covered.

4.4 Discussions

We discuss the validity and generalizability of our framework PAID as below.

As for the validity, our framework may require a large number of user reviews. A small number of reviews might restrict the size and quality of the *Phrase Bank* and further influence the performance



(a) Topic number k.

(b) Penalty μ in issue prioritization.

Figure 4.12: Influence of different parameters on precision of PAID.

of prioritizing issues. Moreover, since the width of "current" in ThemeRiver is calculated by the occurrence of the corresponding issue in the collection, insufficient reviews cause a narrow current, even if the issue would be crucial.

With respect to the generalizability, firstly, the category of apps may affect the phrase-prioritizing process. For game apps, since users tend to leave more significantly shorter and non-informative reviews than the apps of other categories [78], the generated *Phrase Bank* may contain few meaningful phrases. Hence, the selected phrases to topics may not be very helpful for developers. However, to reduce the interference, we can try to use the low-rating reviews and introduce linguistic rules to extract useful features. Secondly, for apps with only one version, dLDA is not applicable. We can use LDA alternatively which possesses the same functionality as dLDA essentially. Finally, we implement and testify PAID on reviews of apps from Google Play Store. It is uncertain whether our framework can achieve similar performance for apps in other stores (*e.g.*, App Store and Amazon Appstore). Future work will be conducted on a more large-scale experimental study to address this threat.

4.5 Summary

This chapter proposes an issue-prioritizing framework PAID to rank app issues automatically and accurately. Chapter 3 has already

shown the effectiveness of tracking user reviews along with time periods. PAID traces the quantitative changes of phrase-level app issues over release versions. Specifically, we propose a rule-based method to extract meaningful phrases, and two-layer filtering method to reduce non-informative reviews. The topics, represented by the most relevant phrases, are tracked along with app versions. The comparison with official changelogs indicates the effectiveness of PAID in scheduling app evolution.

□ **End of chapter.**

Chapter 5

Identifying Emerging Issues based on Online App Review Analysis

Detecting emerging issues (*e.g.*, new bugs) timely and precisely is crucial for developers to update their apps. Although prioritizing issues can help developers schedule app modification, the emerging issues are relevantly more important and urgent. To address this problem, this chapter presents an automated framework named IDEA for identifying emerging app issues effectively based on online review analysis. The main points of this chapter are as follows. (1) It presents the design of an online emerging issue detector IDEA. (2) It introduces an online review analysis method for adaptively determining topics in current versions. (3) It displays large experiments based on real-world datasets and industrial products to verify the effectiveness of IDEA.

5.1 Introduction

App developers are eager to know what is going on with their apps after published [162]. Timely and precisely identifying the emerging issues of apps is of great help for app developers to update their apps, such as fixing bugs, refining existing features, and adding new functions.

The emerging issues detected from user reviews, such as the

existing bugs (*e.g.*, crashes) and unfavorable app features (*e.g.*, too many ads) [72], can provide informative evidence for app developers in maintaining their apps and scheduling the app updates. For example, Facebook Messenger received massive one-star ratings (the lowest rating) in August, 2014, accounting for nearly 94% of all its reviews on Apple's App Store[1], and suffered a large loss of users [10], since the version contained severe privacy issues (*e.g.*, accessing the photos and contact numbers in users' phones). However, such issues had already been flushed out with complaints from over 12,600 user reviews on App Store one month ago. The situation could be effectively alleviated if the emerging issues were timely detected from user reviews. Therefore, user reviews provide an effective and efficient way to identify the emerging issues of apps, which would be a significant help to the developers.

We propose an automated framework IDEA for detecting emerging issues/topics[2] based on online review analysis. IDEA takes reviews of different versions as input. To track the topic variations over versions, a novel method AOLDA (Adaptively Online Latent Dirichlet Allocation) is employed for generating version-sensitive topic distributions. The emerging topics are then identified based on the typical anomaly detection method. To make the topics comprehensible, IDEA labels each topic with the most relevant phrases and sentences based on an effective ranking scheme considering both semantic relevance and user sentiment. The prioritized topic labels are the app issues identified. Finally, IDEA visualizes the variations of app issues along with versions, and highlights the emerging ones for better understanding.

To verify the effectiveness of IDEA, we consider the official app changelogs as ground truth, since they encompass the primary changes of the releases and represent the issues concerned by developers. Our experiments are conducted on six popular apps,

[1]The App Store in this chapter refers to Apple's App Store.

[2]The topics and issues are semantically equal in this chapter.

with two of them from App Store and the others from Google Play. We compare IDEA with the method based on OLDA (Online Latent Dirichlet Allocation) [37], one classical method for emerging issue detection. Results indicate that the average precision, recall, and F-score of IDEA on the subject apps are 60.4%, 60.3%, and 58.5% respectively, which increases the F-score of the OLDA-based method by 72.0%. We also conduct a user survey in Tencent, indicating that 88.9% of respondents think that the identified issues of IDEA can facilitate app development in practice. Moreover, we apply IDEA to four Tencent[3] products which serve hundreds of millions of users worldwide, and confirm the effectiveness and efficiency of IDEA in industrial practice.

The main contributions of this chapter are as follows.

- We propose a framework called IDEA to automatically identify emerging issues from app reviews effectively. Also, IDEA is an online analysis tool and can process new app reviews in a timely fashion.
- We propose a novel method called AOLDA for online review analysis, which adaptively combines the topics of previous versions to generate topic distributions of current versions.
- We visualize the variations of the captured (emerging) app issues along with versions, with the emerging ones highlighted. We publish the code and review data on website[4].
- We verify the effectiveness of IDEA based on the app reviews of six popular apps which are from different categories and platforms. The survey and application in Tencent also validate the performance of our framework in practice.

[3]The company has many popular products, such as WeChat, QQ, and Honor of Kings, and serves billions of users worldwide.

[4]https://github.com/ReMine-Lab/IDEA

5.2 Background and Motivation

In this part, we introduce the concept of emerging app issues, the importance of online review analysis, and our ground truth (*i.e.* the app changelogs).

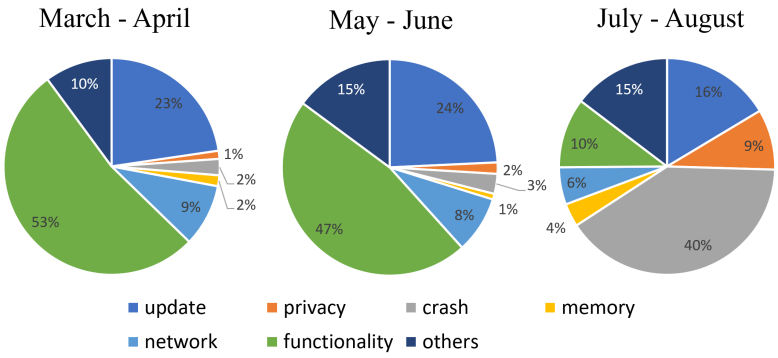
5.2.1 Emerging App Issues

For an app issue to be considered an emerging issue, it must be (heavily) discussed in this time slice but not previously [88]. Figure 5.1 (a) presents the issue distributions of Facebook Messenger in three periods (March-April, May-June, and July-August), based on the manually labeled 100 review samples from each period. Generally, the issue distributions are nearly consistent along with periods, *e.g.*, from March-April to May-June in Figure 5.1 (a). However, emerging issues can influence the issue distribution of one period, creating significant differences with those of previous periods in terms of proportion. For example, the proportion of the crash issue presents a huge increase during the July-August period. We further investigate the number of reviews containing the keyword "crash" along with their timing, and present the results in Figure 5.1 (b). The volume of the crash issue shows a sudden increase around July-August, which signifies that the issue tends to be an emerging issue during that period.

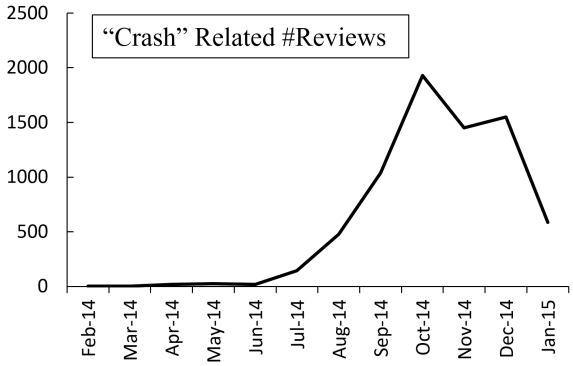
Definition 1 (Emerging Issues in User Reviews) *An issue in a time slice is called an emerging issue if it rarely appears in the previous slice but is mentioned by a significant proportion of user reviews in current slice.*

In Definition 1, the "time slice", the degree of "rarely", and the "significant proportion" can be defined according to different situations. For example, the "time slice" in this chapter corresponds to the app version. Based on the detected emerging issues, developers

can locate the buggy features of their apps efficiently, update the apps accordingly, and ultimately improve the user experience.



(a) Issue distribution in Facebook Messenger.



(b) The number of reviews containing the keyword "crash".

Figure 5.1: Illustration of emerging issues.

5.2.2 Online Review Analysis

Online review analysis (ORA) is an automated way to acquire and process user reviews in real time as reviews are arrived continuously. As shown in Figure 5.2, ORA takes the reviews of slice t (*current review slice*) as input, and outputs analysis results, such as tracking user preference and detecting emerging issues. In this way, the urgent user concerns incarnated by app reviews can be captured by ORA in a timely manner and fed back to developers for instant bug fixing or feature improvement. Thus, ORA is a crucial component in the closed cycle of app development.

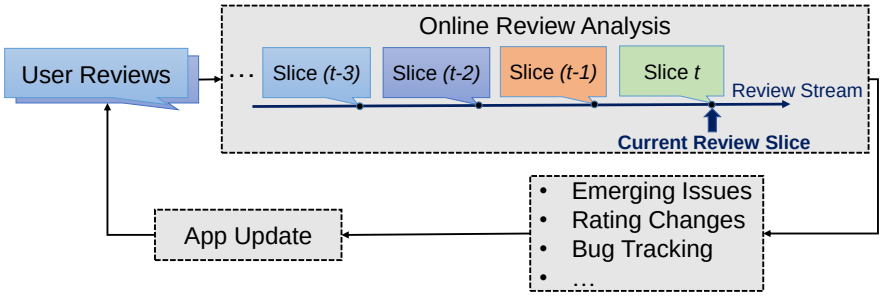


Figure 5.2: Closed cycle for app development.

Currently, most of the app issues mined from user reviews are manually settled or defined [105, 167, 186], such as privacy and GUI, which are usually general categories. Although such definition facilitates the process of task assignment to individuals, it is unfavorable for detecting newly-presented and more detailed issues (*e.g.*, notification center). Thus, for detecting emerging issues, ORA is a practical way due to its timeliness and no need for predefined issues, which has rarely been studied previously.

5.2.3 App Changelogs

App changelogs describe the noticeable modifications of the latest versions for attracting users to install and experience new releases. Similar to user reviews. changelogs also correspond to specific versions. Generally, developers write into the changelogs with information related to whether the apps are adding or removing features, and whether the apps have made improvements with certain devices or to specific bugs. Figure 5.3 illustrates a sample changelog of NOAA Radar Pro, a weather alerts & forecast app in App Store.

As Figure 5.3 indicates, the new version introduces new functionality (*i.e.*, weather reporting) and refines performance issues. The delivered changes exhibit the issues that are concerned by developers. Although the changelogs may not cover all the modifications to the releases, they represent a lower bound and the prominent part of the changes. Hence, changelog is a reasonable ground truth

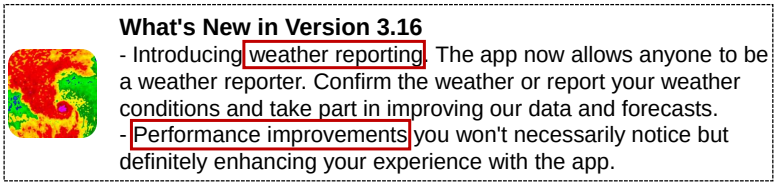


Figure 5.3: Changelog of NOAA Radar Pro. The rectangles highlight two key terms which represent the major changes of Version 3.16.

for verifying whether the extracted emerging issues are helpful for developers.

5.3 Methodology

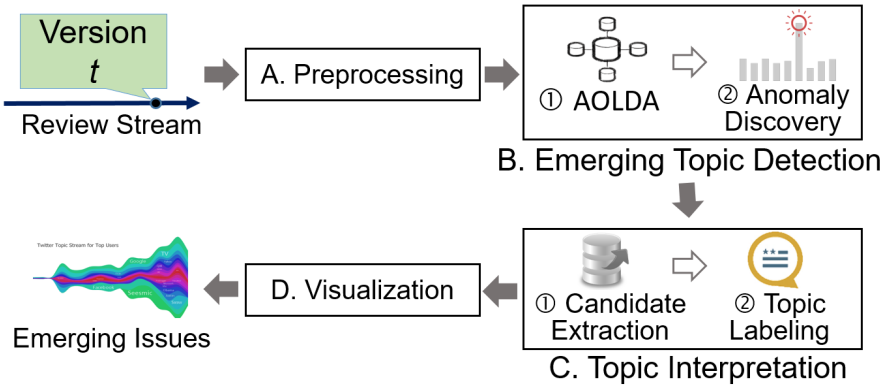


Figure 5.4: Framework of IDEA.

In this part, we first outline the overall framework of IDEA in Figure 5.4 and then elaborate on the four components involved in the framework. Each time, in the first stage (Part A in Figure 5.4), IDEA preprocesses a version of raw reviews from the review stream for reducing noisy words and non-informative words, and extracts phrases for subsequent analysis. In the second stage (Part B in Figure 5.4), the proposed algorithm AOLDA captures the topic distributions of each version by considering the topics in previous versions, based on which emerging topics are identified using anomaly discovery. Then, to interpret the topics (Part C in Figure 5.4), IDEA employs

the meaningful phrases and sentences as candidates to label each topic according to their semantic relevance and user sentiment. The topic labels are the identified app issues. Finally (Part D in Figure 5.4), IDEA visualizes the app issues along with the different versions, and highlight the emerging ones for better understanding.

5.3.1 Preprocessing

Since app reviews are generally submitted via mobile terminals and written using limited keyboards, they contain massive noisy words, such as casual words, repetitive words, misspelled words, and non-informative words (*e.g.*, the words simply describing users' feelings). In the following, we introduce our rule-based methods for formatting words, the phrase extraction process, and our filtering method for reducing non-informative words.

Word Formatting

We first convert all the words in the review collection into lowercase, and then stem each word into its original form. We employ the preprocessing method in [112] for lemmatization. We then replace all digits with "<digit>". Since new terms and casual words would continuously increase in user reviews, we do not employ the dictionaries provided by [186] for avoiding over correction. We adopt the rule-based methods based on [186, 112] to rectify repetitive words, misspelled words, and non-English words.

Phrase Extraction

Since phrases (mainly referring to two consecutive words in this chapter) are employed in Part C of IDEA for interpreting topics, they should be extracted in the preprocessing step and trained along with all the other words in Part B. In this way, we can capture the semantics of each phrase, based on which we can label the topics with

the most relevant phrases. Since the topic labels in phrases should be meaningful and comprehensible, we use a typical phrase extraction method based on PMI (Pointwise Mutual Information) [21], which is effective in identifying meaningful phrases based on co-occurrence frequencies:

$$PMI(w_i, w_j) = \log \frac{p(w_i w_j)}{p(w_i)p(w_j)}, \tag{5.1}$$

where $p(w_i w_j)$ indicates the co-occurrence probability of the phrase $w_i w_j$ and $p(w_i)$ (or $p(w_j)$) represents the probability of the word w_i (or w_j) in the whole review collection. Higher PMI values exhibit that the combination of the two words is more likely to be a meaningful phrase. We extract the meaningful phrases by experimentally set a threshold for PMI. The phrases with PMIs larger than the threshold are extracted.

Filtering

The filtering step aims to reduce the non-informative words, such as emotional words (*e.g.*, "bad" and "nice"), abbreviations (*e.g.*, "asap"), and useless words (*e.g.*, someone). Non-informative words are summarized by two researchers from 1,000 reviews, which are also referred to as *predefined stop words*. The box below lists 18 of the total 78 non-informative words due to space limitations[5]. The *predefined stop words* are filtered out together with the stop words provided by NLTK [17]. We do not employ the supervised method in [52] for filtering, since in this work labeling massive non-informative reviews requires a great deal of manual effort. Finally, all the remaining words and extracted phrases (where the words in each phrase are connected with "_") are fed into the next step for emerging topic detection.

[5]The whole list of predefined stop words can be found in our project website: https://github.com/ReMine-Lab/IDEA.

Predefined Stop Words: *cool, fine, hello, alright, poor, plz, pls, thank, old, new, asap, someone, love, like, bit, annoying, beautiful, dear.*

5.3.2 Emerging Topic Detection

In this section, we aim to detect the emerging topics of current versions by considering the topics in previous versions. We first introduce the proposed method AOLDA for adaptively online topic modeling, from which we capture the topic evolutions along with versions. We then present how we discover the emerging topics (*e.g.*, anomaly topics).

AOLDA - Adaptively Online Latent Dirichlet Allocation

Online Latent Dirichlet Allocation (OLDA) [37] is a classic method for tracking the topic variations of text streams, which models the topics of texts in one time slice based on the topics of the last slice. However, app reviews are typically short and contain massive noise words. Such review features can influence the topic distributions in consecutive versions with OLDA, and thereby decrease the performance of emerging topic detection. To reduce the influence of noise words and more accurately capture the topic evolution along with versions, we propose an adaptively online topic modeling method, AOLDA. The proposed AOLDA improves OLDA by adaptively combining the topic distributions in previous versions. The details are described below.

The preprocessed reviews are divided by version, denoted as $R = \{R^1, R^2, ..., R^t, ...\}$ (where t indicates the t-th version), and input into AOLDA one by one. In AOLDA, each review is treated as one document. The prior distributions over document-topic and topic-word distributions are defined initially, represented as α and β respectively. β determines the topic distributions of the terms in the input. The number of the topics is specified as K. For the k-

th topic, ϕ_k^t is the probability distribution vector over all the input terms. We introduce the parameter - window size w, which defines the number of previous versions to be considered for analyzing the topic distributions of the current version. The overview of the model AOLDA is depicted in Figure 5.5.

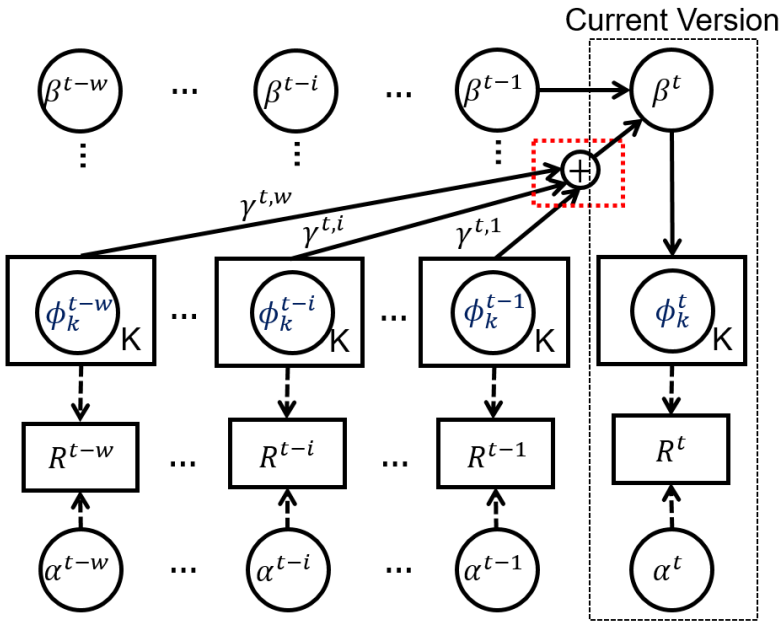


Figure 5.5: Overview of AOLDA. The red rectangle with dashed dots highlights the adaptive integration of the topics of the w previous versions for generating the prior β in the t-th version. R^t is the review corpus in the t-th version. The dotted lines indicate that we simplify the original LDA [50] steps for clearness.

Different from OLDA, as Figure 5.5 shown, we *adaptively* integrate the topic distributions of the previous w versions, denoted as $\{\phi^{t-1}, ..., \phi^{t-i}, ..., \phi^{t-w}\}$, for generating the prior β^t of the t-th version. The *adaptive* integration refers to summing up the topic distributions of different versions with different weights $\gamma^{t,i}$:

$$\beta_k^t = \sum_{i=1}^{w} \gamma_k^{t,i} \phi_k^{t-i}, \tag{5.2}$$

where i denotes the i-th previous version ($1 \leq i \leq w$). The weight $\gamma_k^{t,i}$ is determined by the similarity of the k-th topic between the

$(t-i)$-th version and the $(t-1)$-th version, which is calculated by the softmax function [24]:

$$\gamma_k^{t,i} = \frac{\exp(\phi_k^{t-i} \cdot \beta_k^{t-1})}{\sum_{j=1}^{w} \exp(\phi_k^{t-j} \cdot \beta_k^{t-1})}, \tag{5.3}$$

where the dot product $(\phi_k^{t-i} \cdot \beta_k^{t-1})$ computes the similarity between the topic distribution ϕ_k^{t-i} and the prior of the $(t-1)$-th version β_k^{t-1}. Such adaptive integration can endow the topics of the previous versions with different contributions to the topic distributions of the current version.

Anomaly Discovery

Based on the captured topic evolution by AOLDA, we identify the anomaly topics which present obvious differences with those of the previous versions. The identified anomaly topics are regarded as emerging topics. To obtain the difference of the k-th topics between two consecutive versions, *e.g.*, ϕ_k^t and ϕ_k^{t-1}, we employ the classic Jensen-Shannon (JS) divergence [13]. JS divergence measures the similarity between the two probability distributions:

$$D_{JS}(\phi_k^t || \phi_k^{t-1}) = \frac{1}{2} D_{KL}(\phi_k^t || M) + \frac{1}{2} D_{KL}(\phi_k^{t-1} || M), \tag{5.4}$$

where $M = \frac{1}{2}(\phi_k^t + \phi_k^{t-1})$. The Kullback-Leibler (KL) divergence D_{KL} is utilized to measure the discrimination from one probability distribution P to another Q, computed by:

$$D_{KL}(P||Q) = \sum_i P(i) \log(P(i)/Q(i)), \tag{5.5}$$

where $P(i)$ is the i-th item in P. Higher JS divergence indicates that the two topic distributions have a larger difference.

Based on the computed divergences D_{JS} between the topics of consecutive versions, we capture anomaly topics by leveraging a

typical outlier detection method [156]. The method assumes that the divergences follow a Gaussian distribution with the mean and variance at μ and σ^2 respectively. The anomaly topics are then detected by setting a threshold δ. For the t-th version, the threshold δ^t is dynamically defined according to the following steps.

1. We compute D_{JS} of the previous w versions for each topic, which generates a $w \times K$ matrix (where K is the number of topics).

2. We compute the mean μ and variance σ^2 of all the values in the computed D_{JS} matrix.

3. We set the threshold δ^t as $\delta^t = \mu + 1.25\sigma$, where the coefficient 1.25[6] is experimentally set for accepting 10% of topics as anomaly topics, as shown in Figure 5.6.

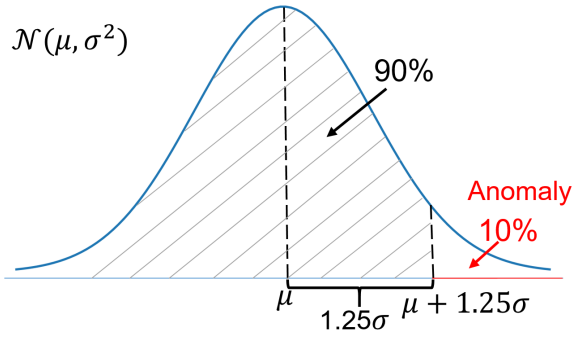


Figure 5.6: Gaussian distribution for anomaly discovery. The shaded area means the integral of the Gaussian distribution, which equals 90%. The topics with divergence larger than δ^t are considered as emerging topics.

For the t-th version, the topics with divergences higher than the defined threshold δ^t are regarded as emerging topics.

[6]The coefficient can be adjusted according to the percentage of anomaly topics to be discovered. We use 1.25 here for accepting 10% of the total topics as anomalies.

5.3.3 Topic Interpretation

The topics based on AOLDA are represented as the probability distributions over all the input terms. One snapshot of the top five terms to each topic is illustrated in Table 5.1. By only observing a few words, it would be non-trivial for developers to capture the meaning of the topics. In this section, we aim to interpret the topics automatically. To interpret each topic, we can utilize words, phrases, sentences, or entire reviews. However, single words may be ambiguous in semantics and cannot display the complete meanings of the topic. For example, we list the top five relevant words for each of the four topics of YouTube, as shown in Table 5.1, although both the words "video" and "work" are most relevant to Topics 2 and 4, these two topics may deliver different meanings, *e.g.*, Topic 2 is related to the video descriptions and Topic 4 is about loading videos. Moreover, one review may complain about several issues. For example, one Instagram user complains about the videos and stories in one review: *Videos don't post. Videos don't load. Stories disappear all the time*. Therefore, topic labels in words or reviews may not be helpful in accurately capturing the semantics of the topics. To render the topics comprehensible, we employ the most relevant phrases and sentences to label each topic in this section.

Candidate Extraction

We obtain candidate phrases and sentences for labeling topics.

Phrase Candidate: The candidates of the phrase labels are generated based on the extracted phrases. *Three rules* are employed to identify more meaningful phrases: 1) Length limit: The length of each word in the phrase should be no less than three; 2) Stop word limit: The phrase should not contain words that are in the stop word list of NLTK [17]; and 3) Part-Of-Speech limit: The phrase should include at least one noun or verb, and no adverbs (*e.g.*, "here") or determiners (*e.g.*, "the").

Table 5.1: Top five terms for each topic of YouTube.

Topic	Topic 1	Topic 2	Topic 3	Topic 4
Term	comment	link	back	load
	say	video	also	video
	reply	open	button	even
	try	work	change	work
	error	description	go back	take

Sentence Candidate: We employ the reviews before the filtering step in, starting by chunking them into sentences based on NLTK's punkt tokenizer [22]. Then we retrieve sentences with more than four words, during which the noisy sentences (such as *so far so bad* and *great one*) are filtered out. The remaining sentences are regarded as our sentence candidates.

Topic Labeling

The topic labeling method is a ranking method, which considers two aspects: the semantic similarity between the candidates and the topics, and also the user sentiment of the candidates.

Semantic Score: Good topic labels should cover the latent meaning of the topic [119]. The semantic score measures the semantic similarity between the candidate and the topic. Moreover, the labels of different topics should be discriminative and cover different aspects of input reviews, instead of delivering overlapping information. Hence, the semantic score of one candidate involves the semantic similarity to the target topic and also the semantic similarities to all the other topics. A good topic label should be similar to the target topic and also different from the other topics in semantics.

We employ the method in [119] to measure the semantic similarity between one phrase candidate a and the target topic ϕ_k^t, defined as:

$$sim(a, \phi_k^t) = -D_{KL}(a||\phi_k^t) \approx \sum_w p(w|\phi_k^t) \log \frac{p(a, w|C)}{p(a|C)p(w|C)}, \tag{5.6}$$

where $p(w|\phi_k^t)$ is the probability of term w in the topic distribution ϕ_k^t. $p(w|C)$ and $p(a|C)$ denote the percentages of the terms w and a in the whole review collection C, respectively. The $p(a, w|C)$ indicates the co-occurrence probability of the two terms a and w in the collection C. For the sentence candidates s, we utilize Equation (5.7) to calculate the similarity.

$$sim(s, \phi_k^t) = -D_{KL}(s||\phi_k^t) \approx \sum_w p(w|\phi_k^t) \log \frac{p(w|s)/len(s)}{p(w|\phi_k^t)}, \tag{5.7}$$

where $p(w|s)$ can be calculated by the term frequency of w in the sentence s. The semantic score is then defined by combining $sim(l, \phi_k^t)$ with the similarity scores to other topics $\sum_{j \neq k} sim(l, \phi_j^t)$, which means the label l should be semantic close to the topic distribution ϕ_k^t and discriminate from other topic distributions.

$$Score_{sem}(l, \phi_k^t) = sim(l, \phi_k^t) - \frac{\mu}{K-1} \sum_{j \neq k} sim(l, \phi_j^t), \tag{5.8}$$

where l can be a phrase candidate a or sentence candidate s, and K is the number of topics. The parameter μ is utilized to adjust the penalty for the semantic similarities to other topics. Larger μ signifies that the candidates that are more different from other topics.

Sentiment Score: The topic labels should reflect users' concerns. Generally, the reviews with low ratings tend to express poor user experience and app issues [52], and the reviews with longer lengths are more likely to provide valuable information to

developers. Therefore, we compute the sentiment score $Score_{sen}$ of one candidate l by combining the user ratings and review lengths:

$$Score_{sen}(l) = \exp(-\frac{r_l}{\log(h_l)}), \tag{5.9}$$

where l can be a phrase candidate or sentence candidate. r and h denote the average user rating and the average word length of the reviews containing l, respectively.

Overall Score: We prioritize the candidates for each topic based on their semantic scores and sentiment scores. The overall score $Score(l, \phi_k^t)$ is defined as:

$$Score(l, \phi_k^t) = Score_{sem}(l, \phi_k^t) + \lambda\, Score_{sen}(l), \tag{5.10}$$

where the weight λ is to balance the two aspects. In this manner, all the topics including the detected emerging topics are labeled with the prioritized candidates. The topic labels are the identified app issues. For each topic, there is a trade-off between the number of prioritized labels and the cost of user comprehension (*e.g.*, too many labels usually spend users more time in understanding the meaning of the topic). According to the survey [188], three labels are the moderate choice for users to comprehend the topics. Therefore, for one topic, we choose the ***three*** most relevant phrases and sentences respectively as labels for each topic.

5.3.4 Visualization

In this part, we visualize the the evolution of app issues (*i.e.*, topic labels) along with versions for better understanding. We employ an *issue river* to display issue variations. Figure 5.7 presents one example of YouTube for iOS. All the app issues constitute one river and each branch of the river indicates one topic. By moving the mouse over one topic, one can track detailed issue changes along

with versions, where the emerging issues are highlighted as shown in Figure 5.7. The app issues with wider branches are of greater concern to users, where the $width$ of the k-th branch in the t-th version is defined as:

$$width_k^t = \sum_a \log Count(a) \times Score_{sen}(a), \tag{5.11}$$

where $Count(a)$ is the count of the phrase label a in the review collection of the t-th version, and $Score_{sen}(a)$ denotes the sentiment score of the label a.

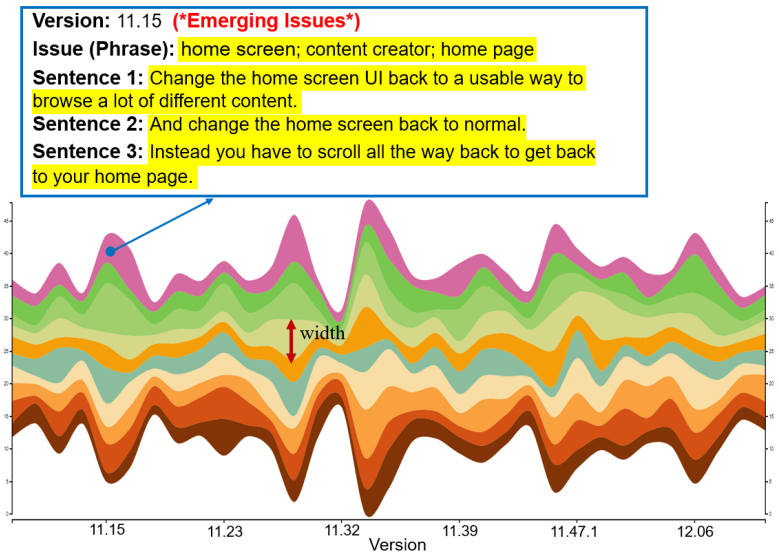


Figure 5.7: Issue River of YouTube for iOS. The number of topics K is set as 10, corresponding to 10 branches of the river. The horizontal axis represents the app versions, and the branches with larger widths indicate that the corresponding issues are more cared about by users at those versions.

5.4 Experimentation

We evaluate the performance of IDEA in identifying emerging app issues based on case studies. In this section, we explain how we select the subject apps for experiments, the performance metrics,

and finally the comparison results of different methods. We focus on answering the following three research questions.
RQ1: What is the performance of IDEA in identifying emerging app issues?
RQ2: Can IDEA achieve better performance compared with other methods?
RQ3: How do different parameter settings impact the performance of IDEA?

5.4.1 Dataset

We select the subject apps based on the following four criteria: The apps are i) popular apps in the app markets - indicating that the developers would update their apps regularly and the user reviews can be collected from several consecutive versions; ii) apps from different categories and platforms - to ensure the generalization of the proposed framework; iii) apps with enough user reviews - which necessitates an automated analysis; and iv) apps with detailed changelogs for most versions - to facilitate our validation process.

To obtain apps that satisfy the first three criteria, we randomly inspect the apps ranked in the top 100 on either App Store or Google Play according to App Annie [2], an app analytics platform. Only the apps with more than 2,000 US reviews [52] are inspected further, since significant effort is required for manual analysis. To filter out the apps that do not meet the fourth criterion, we check the historical changelogs of these apps. We eliminate apps with more than five successive sketchy changelogs, *i.e.*, the changelogs provide no details related to what functionality had been changed or how the user experience was being affected. One example of sketchy changelogs is “Multiple bug fixes and improvements across the entire app”, where the bugs and improvement are not concrete enough for verifying prioritized app issues. Finally, we select six subject apps, with the details illustrated in Table 5.2.

Table 5.2: Subject apps.

App Name	Category	Platform	#Reviews	#Versions
NOAA Radar	Weather	App Store	8,363	16
YouTube	Multimedia	App Store	37,718	33
Viber	Communication	Google Play	17,126	8
Clean Master	Tools	Google Play	44,327	7
Ebay	Shopping	Google Play	35,483	9
Swiftkey	Productivity	Google Play	21,009	16

In Table 5.2, we list the subject apps with the app name, category, platform, the number of reviews crawled, and the number of versions in the review collection. Overall, we obtain 164,026 reviews (from August 2016 to April 2017) for the six apps, from 89 versions in total. The apps are distributed in different categories, with two of them from App Store and the others from Google Play. With multiple categories and platforms, the generalization of IDEA can be ensured.

5.4.2 Performance Metrics

The app changelogs, *i.e.*, our ground truth, are collected from App Annie. Since the prioritized issues of IDEA are in phrases and sentences, we manually extract key terms from these changelogs for verification. One example is illustrated in Table 5.6, with the key terms highlighted. For each key term in changelogs, we validate whether the term is covered by the prioritized issues. Since the *word2vec* model [121] can accurately capture the semantic meanings of input terms based on their vector representations, we obtain the cosine similarities between each key term and the phrase-level issues based on the model. The key term is considered covered if its similarity to one of the issues is larger than 0.6 [92]. For sentence-level issues, we split the sentences into terms (including phrases and words) and verify whether the key term in changelogs can be covered in a similar way. We employ such semi-automatic

evaluation method to facilitate parameter adjustment and comparison with other methods.

We employ three performance metrics[7] for verifying the effectiveness of IDEA. The first metric is for measuring the accuracy in detecting emerging issues, defined as $Precision_E$. The second is to evaluate whether our prioritized app issues (including both emerging and non-emerging issues) reflect the changes mentioned in the changelogs, defined as $Recall_L$. We introduce the third metric F_{hybrid} to measure the balance between $Precision_E$ and $Recall_L$. Higher values of F_{hybrid} indicate that changelogs are more precisely covered by detected emerging issues and more changelogs are reflected in the prioritized issues.

$$\begin{aligned} Precision_E = \frac{I(E \cap G)}{I(E)}, \quad Recall_L = \frac{I(L \cap G)}{I(G)}, \\ F_{hybrid} = 2 \times \frac{Precision_E \times Recall_L}{Precision_E + Recal_L}. \end{aligned} \tag{5.12}$$

where E, G, and L are three sets, containing the detected emerging issues, the key terms in the changelogs, and all app issues (including both emerging and non-emerging issues), respectively. $I(\cdot)$ denotes the number of the issues in $\cdot$. During evaluation, we experimentally set the parameters as $w = 3$, $K = 10$, $\lambda = 0.5$, $PMI = 5$, and $\mu = 0.75$. We also initialize α and β with 0.1 and 0.01 respectively.

5.4.3 Answer to RQ1: Case Study

In this part, we evaluate the performance of IDEA by employing a case study on YouTube. We first present the results of the version-sensitive topic distributions based on AOLDA, then exhibit the

[7]We do not involve $Recall_E$ for validation since changelogs possibly include partial emerging issues. Also, $Precision_L$ cannot be considered because changelogs may cover items other than user-concerned issues. Here, $Precision_E$ and $Recall_L$ measure the precision of the emerging issues and coverage rate of changelogs by all the extracted issues respectively, which are consistent with the standards and convincing for this task.

Table 5.3: Topic-word distributions based on AOLDA.

	v11.07	v11.10	v11.11
Topic 1	link open video work **description**	open video work fine go **click**	video watch go want change
Topic 2	make want button back use	<digit> thing get interface want	back make would button people

prioritized labels to interpret the topics, and finally illustrate the performance of the proposed framework on YouTube.

Result of AOLDA

Table 5.3 depicts the example topic-word distributions based on AOLDA, where the top five words are listed for each topic. According to the table, the general meanings of the topics are consistent along with versions. For example, Topic 1 is related to the video for all the three versions, and Topic 2 is constantly related to the user interface. However, for one topic, the specific meanings may be distinguished in the three versions. Take Topic 1 as an example. The topic may discuss the video description/link for version 11.07, while it talks about "click"-related things in version 11.10. It would be very laborious for developers to comprehend topics based on the top words. Therefore, we conduct automatic topic interpretation in the next step.

Result of Topic Interpretation

Table 5.4 illustrates the prioritized phrases for labeling topics, where only one of the three labels are listed for saving space.

The highlighted labels in Table 5.4 are the emerging app issues detected by the anomaly discovery method. Topic 1 of version 11.07 is interpreted as "description box", which is consistent with the meaning of that topic in Table 5.3 intuitively. Table 5.5 illustrates the ranked sentence for labeling each topic. Although phrase labels can be quickly understood, we discover that sentence labels can detail the information conveyed by phrases and interpret the topics more comprehensively. For example, the sentence label of Topic 1 for version 11.07 (*i.e.*, "...click a link in the description...") provides more details than the corresponding phrase label ("description box") in Table 5.4. With both issues in phrases and sentences, developers can efficiently spot and locate specific app issues. To help developers gain better understanding, we visualize the identified issues along with versions in Figure 5.7. By moving the mouse over Topic 10 of version 11.15, we can observe both phrase-level and sentence-level issues, among which the emerging ones are highlighted. Developers can readily track the changes of each topic and discover urgent issues in a timely manner.

Table 5.4: Topic labels in phrases for YouTube. The highlighted ones indicate detected emerging issues. The value after each label is the overall score of the label.

	v11.07	v11.10	v11.11
Topic 1	description box: 2.03	comment section: 1.48	notification center: 1.33
Topic 2	user interface: 1.25	split screen: 1.23	split screen: 0.94
Topic 3	playback error: 1.44	battery drain: 0.99	performance improvement: 1.41
Topic 4	certain spot: 1.81	cpu usage: 0.85	camera roll: 1.22
Topic 5	profile picture: 2.19	main page: 1.11	home screen: 1.18
Topic 6	say playback error: 1.54	long period: 0.92	force quit: 1.26
Topic 7	copyright issue: 1.11	bring back: 1.14	nothing happen: 1.53
Topic 8	take forever: 1.88	ten minute: 1.12	pure torture: 1.02
Topic 9	sound quality: 1.55	major issue: 1.45	buffer forever: 1.03
Topic 10	home button: 1.15	full screen: 1.07	home page: 1.29

Table 5.5: Topic labels in sentences for YouTube. The highlighted ones are the detected emerging issues.

	v11.07	v11.10
Topic 1	I mean it work but why do you take off where you would click a link in the description and it doesn't even let me go through the video: -0.05	It say error every time I try to reply back to a comment: 0.52
Topic 2	But right now the lack of multitasking have actually make it a better experience to use YouTube in safari: -0.79	Add split view and slide over but no picture in picture: -1.36
Topic 3	Please fix this app fix this bug and that playback error: -0.80	Dear YouTube please release a fix for overheat issue on older iPhone and the battery drain just too ridiculous: -0.45

Performance Evaluation

We collect the ground truth of YouTube based on the method. Table 5.6 displays part of the changelogs. We manually inspect whether the identified app issues of one version can be reflected in the changelogs of the next version. According to Table 5.6, version 11.10 improves the user interface by adding the functionality of multitasking (*i.e.*, sliding over and splitting view [16]) and fixes the bug in video descriptions. Referring to Table 5.4 and Table 5.5, we discover that the two issues are detected by IDEA in Topic 1 and Topic 2 of the previous version 11.07. Then to statistically measure the performance of our framework, we employ the proposed three metrics. Based on the collected 33 versions for YouTube, IDEA achieves $Precision_E$, $Recall_L$, and F_{hybrid} at 0.628, 0.666, 0.636 in sentence-level issues and 0.592, 0.472, and 0.523 in phrase-level issues, respectively.

Discussion of the performance: Since the changelogs may not cover all the changes in releases, the metric $Precision_E$ represents a lower bound of the performance. For example, the highlighted emerging issues, such as "split screen" and "battery drain" for version 11.10 in Table 5.4, are not clearly embodied by the changelog of version 11.11 (shown in Table 5.6). We then inspect the reason why the detected issues "fail" to be noticed by developers. We discover

Table 5.6: Changelog of YouTube

Version	Date	Changelog
11.10	22-Mar-16	(1) Added slide over and split view support (2) Moved home tabs into navigation bar for iPad in landscape mode (3) Fixed bug that prevented URLs in video descriptions from opening
11.11	29-Mar-16	(1) Fixed bug where accessibility VoiceOver looped over the same elements (2) Fixed issue where the video couldn't be exited after completing (3) Bug fixes and stability improvements

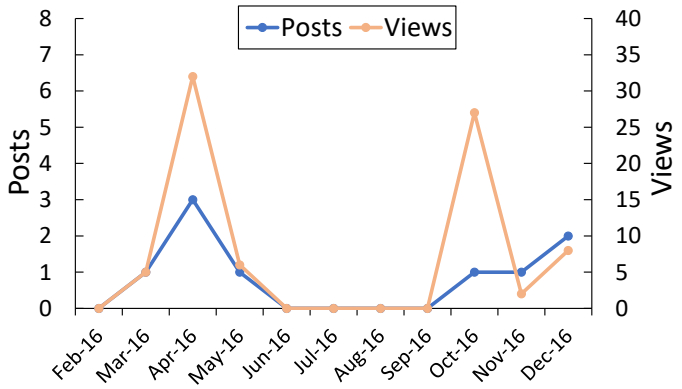


Figure 5.8: Count of posts and views related to the battery issue in YouTube iOS forum.

that "split screen" is one new added feature of version 11.10 and it is reasonable for a hot discussion about the drawbacks of this feature in the user reviews, which explains why "split view" is identified as one emerging issue. Then for the issue "battery drain", we dig into the official user forum of YouTube for iOS [28], and observe the number of posts and views of the issue by searching the phrase (illustrated in Figure 5.8). We find that there exists a sudden increase in the counts of posts and views around May 2016, which also demonstrates that the battery issue was an emerging issue for the version. Therefore, we summarize that changelogs may not completely cover all emerging issues, and our performance metric computes a lower bound of the performance of IDEA. The comparison with other methods can validate our proposed framework more sufficiently.

5.4.4 Answer to RQ2: Comparison Results with Different Methods

For validating the performance of AOLDA in IDEA, we choose the typical method for online topic modeling - OLDA [37]. For evaluating the proposed topic labeling method, we also compare with the method only considering the sentiment score for labeling (denoted as IDEA-R), and the method only considering the semantic score for labeling (denoted as IDEA-S). For clarity, our proposed framework is represented as IDEA$^+$. Table 5.7 illustrates the comparison results on the six subject apps. We discuss the performance of IDEA from three aspects in the following subsections.

Issues in Phrases v.s. Issues in Sentences

According to the results of IDEA$^+$ in Table 5.7, issues in sentences can attain better performance than those in phrases by 30.7%, 52.5%, and 43.2% in $Precision_E$, $Recall_L$, and F_{hybrid} on average respectively. This may be attributed to the fact that sentences can deliver more detailed and complete information than phrases, and

Table 5.7: Comparison result of different methods on six subject apps. The value under each app name indicates the average number of reviews across the versions.

App Name (#avg. reviews)	Method	Phrase			Sentence		
		$Precision_E$	$Recall_L$	F_{hybrid}	$Precision_E$	$Recall_L$	F_{hybrid}
NOAA Radar (523)	OLDA	0.468	0.528	0.473	**0.482**	0.622	0.534
	IDEA-R	**0.606**	0.461	0.520	0.478	0.570	0.503
	IDEA-S	0.250	**0.530**	0.340	0.417	0.547	0.473
	IDEA$^+$	0.571	0.497	**0.531**	0.476	**0.639**	**0.546**
Youtube (1,143)	OLDA	0.441	0.462	0.451	0.578	0.664	0.597
	IDEA-R	0.506	0.429	0.456	0.550	0.659	0.586
	IDEA-S	0.548	0.466	0.502	0.456	0.656	0.522
	IDEA$^+$	**0.592**	**0.472**	**0.523**	**0.628**	**0.666**	**0.636**
Viber (2,141)	OLDA	0.157	0.305	0.166	0.313	0.550	0.375
	IDEA-R	0.542	0.326	0.407	**0.625**	0.571	0.597
	IDEA-S	0.500	**0.342**	0.406	0.500	0.518	0.509
	IDEA$^+$	**0.625**	0.340	**0.440**	**0.625**	**0.651**	**0.638**
Clean Master (6,332)	OLDA	0.300	0.269	0.160	0.200	0.421	0.129
	IDEA-R	0.500	0.216	0.301	**0.750**	0.377	0.502
	IDEA-S	0.067	0.289	0.366	0.500	0.398	0.443
	IDEA$^+$	**0.667**	**0.318**	**0.431**	0.667	**0.434**	**0.526**
Ebay (3,943)	OLDA	0.167	0.238	0.196	0.500	0.488	0.494
	IDEA-R	**0.229**	0.243	0.220	**0.646**	0.496	0.561
	IDEA-S	0.125	**0.285**	0.132	0.354	0.476	0.406
	IDEA$^+$	**0.229**	0.251	**0.227**	**0.646**	**0.527**	**0.580**
SwiftKey (1,313)	OLDA	0.100	0.567	0.148	0.367	0.617	0.458
	IDEA-R	0.333	0.611	0.376	0.417	**0.733**	0.515
	IDEA-S	0.333	0.622	0.372	0.500	0.711	**0.587**
	IDEA$^+$	**0.517**	**0.653**	**0.523**	**0.583**	0.700	**0.587**

thereby cover more key terms in changelogs. Focusing on the metric F_{hybrid}, employing sentence-level issues improves the properties of using phrase-level issues by 2.7%∼1.56 times. For $Precision_E$, the issues in sentences increase those in phrases by -16.7%∼1.8 times. The negative increase only occurs to the app NOAA Radar, which may be because the small datasets of the app (512 reviews per version) introduce instability for our framework [14]. For the metric $Recall_L$, IDEA$^+$ shows an increase range of 7.1%∼1.1 times. Overall, sentence-level issues can better represent app issues, and we employ such issue representations for comparing with different methods in the following.

AOLDA v.s. OLDA

On average, IDEA$^+$ achieves 0.604, 0.603, and 0.585 for $Precision_E$, $Recall_L$, and F_{hybrid} respectively, while the OLDA-based method only obtains 0.407, 0.560, and 0.431 for the three metrics. Considering the metric F_{hybrid}, AOLDA enhances the performance of OLDA by 2.1%∼3.08 times, where OLDA presents the poorest performance (0.129) on the app with the largest quantity of reviews (*e.g.*, Clean Master with 6,332 reviews per version). For the metrics $Precision_E$ and $Recall_L$, our framework can improve the performance by -1.1%∼2.33 times and 0.3%∼18.4% respectively. Although IDEA$^+$ exhibits a slightly lower $Precision_E$ than the OLDA-based method for the app NOAA Radar, it shows better performance in both F_{hybrid} and $Recall_L$, which indicates that our framework can well balance the precision and recall in issue detection.

IDEA v.s. Different Topic Labeling Methods

We discover that IDEA$^+$ can achieve better performance than IDEA-R and present the increase rates at 7.1%, 7.3%, and 7.7% on average for the three metrics respectively. For F_{hybrid}, our framework

improves IDEA-R by 3.4%∼14.0%. When compared with IDEA-S, our framework increases by 34.9%, 10.4%, and 20.7% on average in $Precision_E$, $Recall_L$, and F_{hybrid}, respectively. Therefore, both the user sentiment and semantic similarity should be considered for topic labeling.

5.4.5 Answer to RQ3: Effect of Different Parameter Settings

In this part, we demonstrate the impact of different parameter settings on the performance of our framework. We focus on analyzing two important parameters, including the window size w and the number of topics K. We also explain how we choose the parameters in our experiments.

Window Size

Figure 5.9 illustrates the results of different window sizes on two apps, including YouTube and Ebay. For both apps, the values of F_{hybrid} present an inverted "U" shape for both phrase-level and sentence-level issues. We attribute this to the reason that the topic distributions of the current version are strongly dependent on those of the previous versions. When the window size is set relatively small, the detected issues of current versions may be more divergent and unstable. However, larger window sizes may weaken the distinction of app issues among versions, which is unfavorable for detecting the emerging issues. Since $w = 3$ can achieve the best performance on our datasets (indicated in Figure 5.9), we set the window size as three in our experiments.

The Number of Topics

Generally, the topic number should be defined according to the size of the review collection [42]. In IDEA, a larger topic number can bring more prioritized app issues, which can cover more changelogs

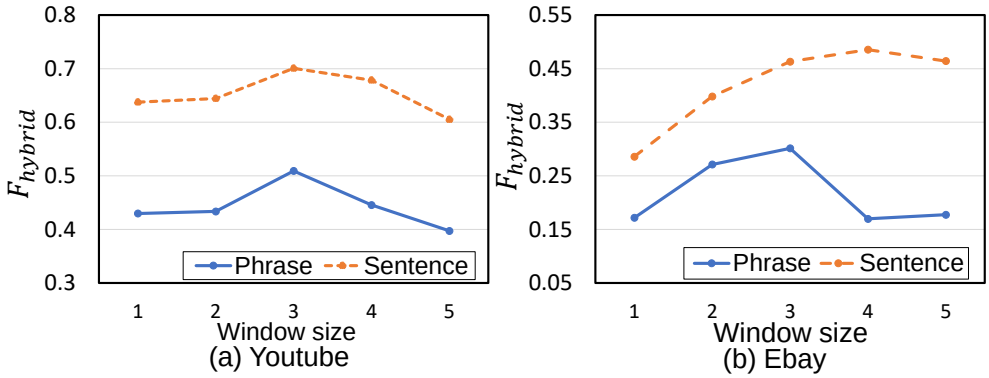


Figure 5.9: Impact of window size.

(*i.e.*, increasing $Recall_L$). However, more app issues may be a double-edged sword, since the metric $Precision_E$ can be decreased. Figure 5.10 shows the results of different topic numbers on two apps, including NOAA Radar and Ebay. For Ebay (on average 3,943 reviews per version), the values of F_{hybrid} display an ascending tendency in both phrase-level and sentence-level issues. But for NOAA Radar (on average 523 reviews per version), a larger topic number will reduce the performance when using phrase-level issues. To better balance the precision and recall, we set the topic number as 10 during experiments.

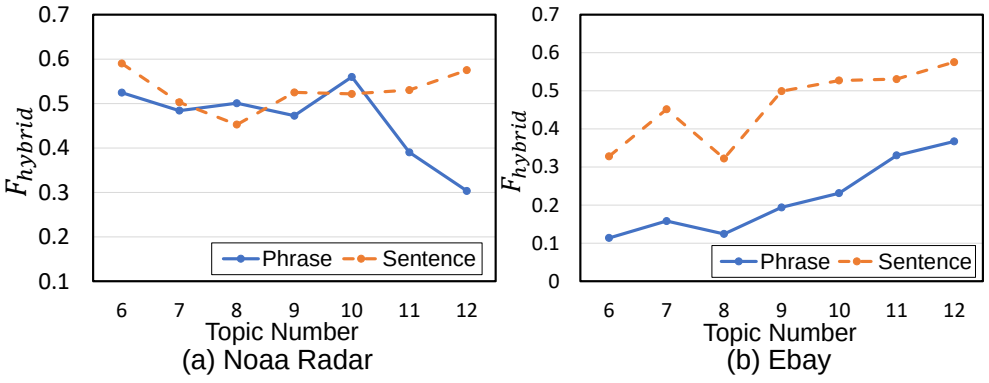


Figure 5.10: Impact of topic number.

5.5 IDEA in Practice

In this section, we explore the performance of IDEA in practice. First, we introduce a user survey conducted in Tencent. Then we describe the successful application of IDEA in Tencent's products.

5.5.1 User Survey

To further demonstrate the significance and effectiveness of our work, we conduct a user study among 45 staff in Tencent, with 29 developers (64.4%), five data analysts (11.1%), four product managers (8.9%), three maintenance engineers (6.7%), one test engineer (2.2%), and three from other positions (6.7%). The user study is conducted through an online questionnaire, which consists of six questions: one question on participants' background, four questions for experimental assessment, and one question for understanding their attitude towards such automatic analysis.

Changelogs as Ground Truth.

We interview the participants about their opinions of using changelogs as ground truth, since changelogs may only include partial changes of the releases. The survey results indicate that 31 (68.9%) of the interviewees agree that changelogs can reflect modified issues of the new releases, and 10 (22.2%) of them indicate a strong approval. Moreover, 88.9% of participants think that changelogs embody the user concerns of the previous releases, with 11.1% echoing strong agreement. Since our framework aims to prioritize app issues based on user reviews, using changelogs as ground truth is reasonable.

Effectiveness of Our Framework.

During the survey, we validate our framework in terms of three aspects: the presentation style of IDEA's results, the performance

achieved by our framework, and the significance of such automatic analysis. The survey results indicate that 75.6% of participants think the visualization with an *issue river* is comprehensible for them, while the phrase-level issues (only with the approval rate at 11.1%) are considered more difficult to understand than sentence-level issues (with an approval rate of 37.8%). For inquiring about their opinions of the performance of IDEA, we present the example results of WizNote [31] with $Precision_E$ and F_{hybrid} at 50%$\sim$60%. According to the survey, 88.9% of the interviewees think that the performance is acceptable in practical usage, and 31.1% strongly approve of such performance. In addition, all the participants think such automatic analysis of detecting emerging issues is significant for app development, with 73.3% of them strongly agreeing with this sentiment. These results provide strong evidence of the effectiveness of our framework.

5.5.2 Successful Story in Industrial Practice

Team X of Tencent aims to provide developers with abnormal events report and operation statistics of 20+ apps of Tencent. Traditional review analysis in X requires lots of manpower. With the increasing quantities of app reviews and the onslaught of spam in user reviews, X has been seeking a means of automatic analysis. We have successfully applied IDEA into X to maintain four apps with review quantities at 500$\sim$5,000 per day. The four apps serve hundreds of millions of users worldwide, and their quality is very important for the company. IDEA obtains user reviews by the hour or day based on the review collection API provided by X. The collected reviews are grouped by versions and processed in real time. The detected emerging issues are fed back to developers for further analysis.

In July of 2017, App Y encountered a serious problem when the content search service was not available for a period of time, and received a sudden increase in the amount of user feedback. With

IDEA, the team X quickly identified the issue and reported it to the development team. The team also confirmed this issue.

Moreover, IDEA can efficiently analyze large numbers of reviews. We deploy IDEA on a PC with Intel(R) Xeon E5-2620v2 CPU (2.10 GHz, 6 cores) and 16GB RAM. For 36,000 product reviews per version, IDEA achieves a high throughput (nearly 160 reviews per second), and only consumes 1.02GB of memory on average. Overall, IDEA is proved to be effective and efficient in quickly pinpointing urgent app issues for developers in the industrial practice.

5.6 Threats to Validity

First, we only select six subject apps for validating our framework and the apps represent a tiny portion of all apps on app markets. Since we utilize user reviews for detecting emerging issues, our methods can be easily applied to other apps, even those with other languages. Also, we alleviate this threat by choosing the apps from different categories and platforms. Second, the number of user reviews can impact the performance of IDEA. However, since small datasets can be easily analyzed manually, our framework aims for automatic analysis of large review datasets. We also mitigate this threat by selecting apps with different quantities of user reviews (on average 523$\sim$6,332 reviews per version). Third, the topic number should be manually defined, which can influence the performance of our framework. Such a threat stems from the original topic modeling method [50], which is still a great challenge in academia [200]. In this chapter, we alleviate this threat by testing on different topic numbers (introduced in the parameter study part). In practice, we can employ heuristic approaches [200] to determine the optimal topic number.

5.7 Summary

Timely and effectively detecting app issues is crucial for app developers. We propose IDEA, a novel framework for automatically identifying emerging issues from user reviews. Our framework can be easily applied to text-based online detection tasks and report emerging issues timely. The industrial practice also validates the effectiveness of IDEA. In the future, we will refine IDEA to be capable of defining the topic number automatically, and make IDEA a distributed algorithm for supporting ultra-large-scale datasets.

□ **End of chapter.**

Chapter 6

Understanding Cross-Platform App Issues

App developers generally publish apps on different platforms, such as Google Play, App Store, and Windows Store, to maximize the user volumes and potential revenue. Due to the different characteristics of the platforms and different user preference, app testing cases on these three platforms should be designed accordingly. In this chapter, we understand the differences in the app issues on these platforms to facilitate app testing process. The main points of this chapter are as follows. (1) It proposes a retrieval method to extract issue-related keywords. (2) It shows the differences and similarities of app issue distributions on different app platforms. (3) It illustrates the effectiveness of prioritizing issues of different platforms.

6.1 Introduction

Smartphones have penetrated into people's daily life. By 2015, the global user volume of smartphones has exceeded half the world's population [19]. Accounting for this popularity is the growing creation and usage of mobile applications (*i.e.*, apps). To distribute the apps to users, developers are required to publish the apps on the distribution platforms specific to mobile apps. Generally, app developers choose to deliver their apps on more than one platforms

to enlarge the potential user volume and revenue [27]. In 2017, the three largest global platforms for app distribution are Google Play, App Store, and Windows Store, which occupy 85.0%, 14.7%, and 0.1% of the market, respectively [23]. These three platforms are then regarded as the focus of our study.

To ensure user experience, developers should examine the software reliability before the app delivery. The unique characteristics of the operating systems indicate that the testing on these platforms is not exactly the same [20], shown in Figure 6.1. For example, Android is more customizable and offers an open platform, while iOS prioritizes the user interface over just about anything [12]. Furthermore, the users of different platforms possess different preferences. For example, iOS users are considered to be more "addicted" to digital devices than Android users [30]. Therefore, different platforms may generate different app issues, and understanding the differences facilitates the app development process for developers.

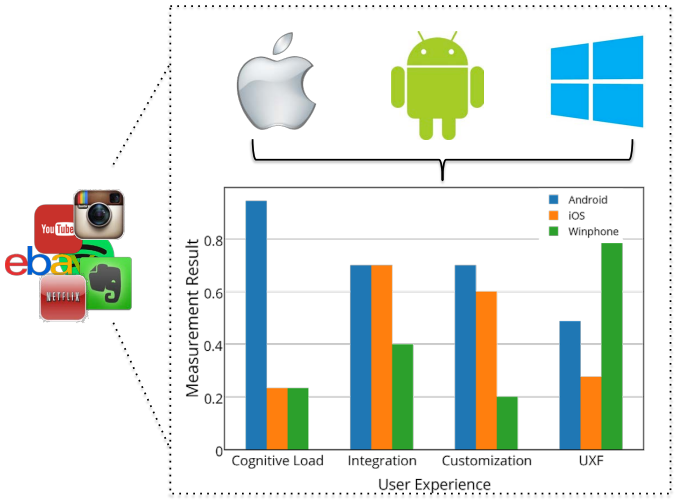


Figure 6.1: User experience on different platforms. Here, "UXF" denotes the user experience friction - the aspects of a device that can annoy users in a niggling way.

The existing studies concentrate on comparing the characteristics of the operating systems, such as the accessing Internet streaming services [106], security mechanisms [36], and the demograph-

ics [44]. There is no exploration of the differences of app issues on these platforms. Although one piece of work [202] analyzes bugs and bug-fixing for projects on different platforms, the work focuses on the updating rates and bug details. To fill this significant gap, in this chapter, we aim at comparing the app issues for the platforms and provide developers with insights on testing apps.

Since user reviews provide a valuable data source for developers to identify potential issues of their mobile apps, we employ the reviews to discover app issues. We have crawled about five million user reviews of 20 apps for the three platforms (*i.e.*, Google Play, App Store, and Windows Store). By examining the app reviews, we choose seven issues for comparison. They are "crash", "battery drainage", "memory consumption", "network connection", "privacy", "spam", and "UI design". We design a framework named CrossMiner to comprehend the issues distributions on these platforms, and provide developers with crucial issues for different platforms.

To analyze the differences of app issues on different platforms, we propose an issue-retrieval method to extract relevant words for each issue. Specifically, we first preprocess raw reviews to obtain input for word2vec[121] model and convert each word into a vector. Then, we cluster the words by k-means algorithm and summarize the corresponding keywords for each issue based on cosine similarity method. By using the issue-retrieval method, we compute and visualize the distribution of each issue on different platforms. For better understanding specific issues, we also prioritize important reviews correspondingly to developers. We conduct an ***empirical study*** on a large scale dataset (4,663,316 reviews of 20 popular apps), and demonstrate the differences and similarities existing along with the three platforms. We also show a case study to verify the effectiveness of CrossMiner in reflecting the important user concerns.

The main contributions of this chapter are as follows:

- We propose an issue-retrieval framework CrossMiner to extract issue-related keywords comprehensively from user reviews.
- We obtain insights about differences and similarities of app issue distributions on different platforms from users' perspective.
- We demonstrate that our framework reflects the importance of user concerns accurately. Developers can also analyze the detailed concerns based on the prioritized user reviews.

6.2 Motivation and Background

A report from [19] illustrates the quantities of apps available for downloading in leading app stores during July 2015. There are more than 1.6 million, 1.5 million, and 0.34 million in Google Play, App Store, and Windows Store, respectively. As a process for improving app's functionality, usability, and consistency, mobile app testing determines the delivery quality to end users, and becomes increasingly important for any companies that desire to keep competitive in the intensive app markets.

However, designing comprehensive app testing cases is time-consuming and sometimes difficult for app developers. One key challenge for the app testing is attributed to the diverse mobile platforms, such as Android, iOS, and Windows Phone. Each mobile operating system possesses unique limitations and properties. App testing across different platforms requires app developers to be familiar with the characteristics of each platform, and design test cases specifically. Moreover, users of different platforms embody different preferences and perceptions about an app [20]. Therefore, comprehending app issues on different platforms can facilitate the whole process for developers.

User reviews can be regarded as the "voices of users". They directly reflect the user experience [134]. Since analyzing user reviews assists developers in fixing bugs and adding new features, different user concerns on different platforms can be captured by

utilizing the corresponding app reviews. Thus, developers can test apps more specifically and efficiently based on the extracted user issues.

In this chapter, we select seven issues which are crucial for app testing [29, 11]. They are "battery", "crash", "memory", "network", "privacy", "spam", and "UI". To verify whether users concern these issues practically, we take Facebook as an example and examine the corresponding user reviews. Table 6.1 illustrates example user reviews regarding these issues.

Table 6.1: Example user review related to each issue type from the Facebook app

Issue	Platform	Review	Rating
Battery	Google Play	This app is the main reason to drain down the battery!	1.0
	App Store	Nice but make wasteful battery, first fix dong.	4.0
	Windows Store	Battery draining app.	1.0
Crash	Google Play	The app crash as soon as i tap on the facebook icon.	1.0
	App Store	Crash and hang issue in ios.... Pls fix.	1.0
	Windows Store	Turrible, it crashes every 4 minutes and its just.	1.0
Memory	Google Play	The app is good but it takes too much memory space.	4.0
	App Store	Since the last update covers much memory space.	3.0
	Windows Store	This it takes whole space in my memory card.	1.0
Network	Google Play	It always gives me a network problem.	1.0
	App Store	Network connection error.	1.0
	Windows Store	Waiting for network for days, slowest app ever.	2.0
Privacy	Google Play	The Big Brother version. No privacy anymore.	1.0
	App Store	Poor. Privacy invading.	1.0
	Windows Store	It got privacy problems.	1.0
Spam	Google Play	Uses too many resources, and includes a lot of spam.	2.0
	App Store	All this spam and posts I didn't make are annoying.	2.0
	Windows Store	This app puts spam ads for weight loss on the news feed.	1.0
UI	Google Play	Change ui of app. its boring to use same ui app.	2.0
	App Store	This app can be so much better...yet the UI just drives me nuts.	1.0
	Windows Store	We need the call feature and little tweak in the UI.	1.0

Table 6.1 demonstrates that users really complain about these issues on different platforms. The highlighted phrases indicate the keywords or key phrases representative for the issues. For example,

with respect to "battery", the users state the issues with "drain down the battery", "wasteful battery", and "battery draining" on the three platforms respectively. Furthermore, we discover that almost all the reviews are corresponding to low ratings (less than three). Hence, we suppose that these issues are concerned by users. Investigating the issues on different platforms is helpful for app developers.

In this chapter, we aim at implementing a framework, namely CrossMiner, to help developers understand the differences of app issues on different platforms based on user reviews. Developers can then focus on the important issues on these platforms during the app testing. Given the app reviews of each platform, CrossMiner automatically prioritizes the issues on this platform. We focus on the seven app issues shown in Table 6.1.

6.3 Methodology

This section first gives an overview of the proposed tool, CrossMiner, and then elaborate on each of the two major procedures in CrossMiner, including clean review extraction and keywords generation.

6.3.1 Overview of CrossMiner

Figure 6.2 illustrates the overview of the proposed framework CrossMiner, which consists of three steps. The first step preprocesses and filters raw user reviews from the three app stores, including Google Play, App Store, and Windows Store. In this process, raw user reviews are converted into clean user reviews to facilitate the following steps. The second step trains a model for our dataset. This model can extract the keywords automatically for the seven issues illustrated in Table 6.1. Based on the extracted keywords, we prioritize these issues for each platform, and compare these issues distributions among the three platforms. To gain an in-depth

understanding of specific issues, we also recommend essential user reviews corresponding to these issues. Finally, we visualize the experimental findings for app developers, and will be detailed in the experimental part.

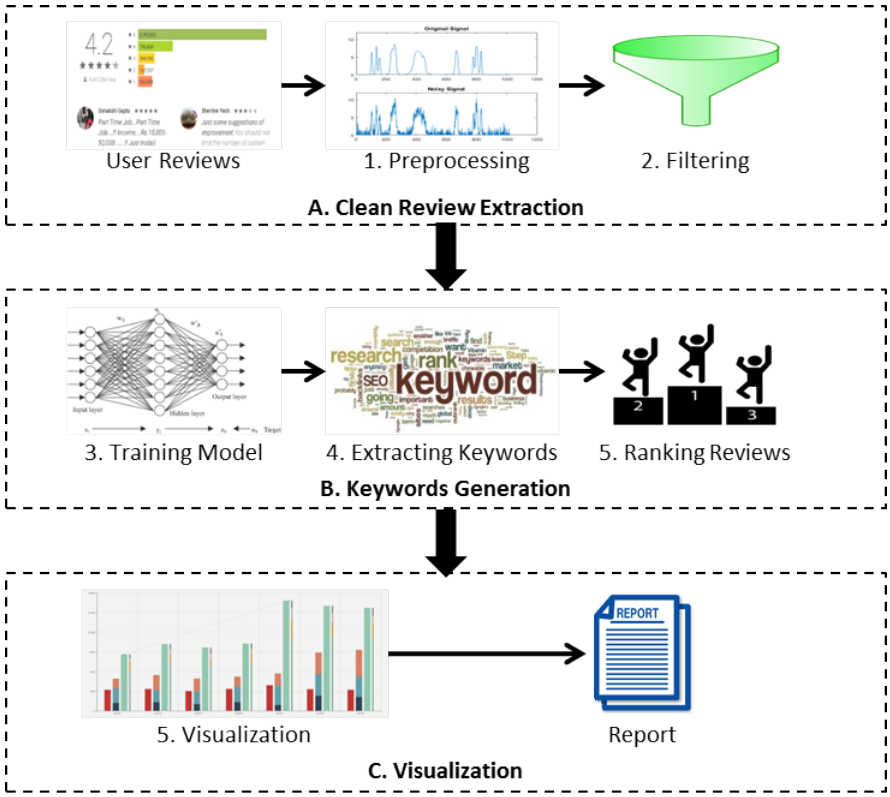


Figure 6.2: Overview of the framework CrossMiner

6.3.2 Step 1: Clean Review Extraction

App reviews are short in length, and contain massive misspelled words and made-up words [52]. In the first step, CrossMiner obtains clean reviews to facilitate the model training in the subsequent process. The step mainly contains two parts, *i.e.*, preprocessing and filtering.

Preprocessing

To facilitate the subsequent analysis, we first remove the non-English characters existing in the raw reviews and convert raw reviews into lowercases. Then we remove all the non-alpha-numeric symbols but keep the punctuations to ensure the semantic integrity [5]. Finally, we tokenize the reviews to word-level collections. To better reduce the inflectional forms to the common base forms, we propose a novel lemmatization method. We do not utilize stemming, since it usually refers to a crude heuristic process that chops off the ends of words, illustrated in Table 6.2. Some words are difficult to identify after stemming (*e.g.*, "minutes" to "minut", and "adding" to "ad").

Table 6.2: Results of stemming and lemmatization

Original Word	Stemming	Lemmatization(v)	Lemmatization(n)
are	be	be	are
adding	ad	add	adding
several	sever	several	several
settings	set	settings	setting
developers	develop	developers	developer
minutes	minut	minutes	minute
serves	serv	serve	serf
does	doe	do	doe
uses	use	use	us
pass	pass	pass	pas
less	less	less	le

Furthermore, considering the influence of the part of speech, we combine the lemmatization for verbs, denoted as Lemmatization(v), and the lemmatization for nouns, denoted as Lemmatization(n). Lemmatization(n) cannot convert verbs into the base forms. Moreover, some words are converted into other words that are totally irrelevant with original words, such as "serves" to "serf", "does" to "doe", "uses" to "us", which can be compensated by Lemmatization(v). However, Lemmatization(v) cannot achieve the desired result either. For example, "settings" and "developers" keep unchanged after the lemmatization(v), while Lemmatization(n) can

return the correct forms. Therefore, neither lemmatizations can achieve ideal results solely.

The combinations of Lemmatization(v) with Lemmatization(n) are implemented as following. We first lemmatize all words by Lemmatization(v). We then conduct the Lemmatization(n) for words without "ss" ends, since Lemmatization(n) converts the words ending with 'ss' into other words instead of their base forms (*e.g.*, "pass" to "pas", and "less" to "le", illustrated in Table 6.2). Table 6.3 presents the results of our proposed lemmatization method, which demonstrates its effectiveness. The Lemmatizer employed is implemented based on the NLTK [17].

Table 6.3: Results of proposed lemmatization method

Original Word	Proposed Lemmatization
adding	add
several	several
settings	setting
developers	developer
minutes	minute
serves	serve
does	do
uses	use
pass	pass
less	less

Filtering

The previous step generates a preprocessed review collection, with examples presented in Table 6.4. We then classify each review into three types, *i.e.*, "useless", "non-informative", and "informative". The "useless" reviews are those reviews with too much made-up or misspelled words. Some users type letters just arbitrarily during the review writing, which cannot provide any suggestions to developers. The "non-informative" reviews contain no information beneficial for the app development (*e.g.*, "nice app.", and "pls fix it!"). We retain the "non-informative" reviews since they possess intact sentence

structures, which can serve as the input of the model training. All the other reviews are determined as "informative" reviews, which offer developers suggestions on fixing bugs or adding features. In the end, only the "useless" reviews are filtered out for the subsequent process.

Subsequently, to filter noises in the reviews, we conduct a rule-based method in the **word-level** granularity and spell checking at the **review-level** granularity.

a) Word-Level: Three rules are adopted during the word-level filtering process, illustrated in the following.

Rule 1 (Consecutive Duplicate Letter Limit). *We remove consecutive duplicates, since the length of consecutive repeated letters is less than three generally [7]. Specifically, if the repetition times of a letter is more than two, the repeated ones will be eliminated (e.g., "suuuuper" to 'super").*

Rule 2 (Word Length Limit). *We remove all the words whose length is more than 15, since 99.93% English words' lengths are less than 16 [9] (e.g., "jfieendkwjjfkkdn").*

Rule 3 (Consecutive Duplicate Word Limit). *We remove consecutive duplicate words in a sentence (e.g., "very very very beautiful' to "very beautiful"').*

b) Review-Level: In review level, we employ Enchant [8], a generic spell checking library, to conduct the spell checking in each review. Any reviews with more than half words not correctly spelled will be removed.

After preprocessing and filtering, we convert all raw reviews into clean reviews. Table 6.4 presents the results after preprocessing and filtering.

6.3.3 Step 2: Keywords Generation

We have obtained clean reviews based on preprocessing and filtering in the last step. In this step, we train the word2vec model on our

Table 6.4: Example reviews after preprocessing and filtering.

Type	Preprocessed Review	Clean Review
Useless	gk bgitu jlek anyway. tp gmna lg mw. hrus ttap there perubhan.	
Non-informative	nice app.	nice app.
Non-informative	pls fix it!	pls fix it!
Non-informative	it be suuuuper.	it be super.
Non-informative	jfieendkwjjfkkdn i dont know what to say its awsone.	i dont know what to say its awsone.
Non-informative	very very very beautiful.	very beautiful.
Informative	it be so slow and it glitch up.	it be so slow and it glitch up.

datasets, based on which the keywords are retrieved with respect to the seven issues, *i.e.*, crash, battery, memory, network, privacy, spam, and UI. Finally, reviews for each issue are prioritized according to its importance and usefulness to developers.

Training Model

To establish the model, we first convert all words to vectors by employing word2vec [121], a neural network implementation for learning vector representations of words. Single sentence serves as the input of word2vec, generally represented by a list of words. Since reviews may consist of several sentences, we demand to split the reviews into sentences. Here, NLTK's punkt tokenizer [22] is employed for the splitting. Based on the obtained parsed sentences, we then adopt skip-gram, one flavor of word2vec, as our training model.

Given a sequence of words to train $\{w_1, w_2, w_3, ..., w_T\}$, the objective function of skip-gram model is to maximize the log probability of any context word given the current center word, defined as

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j}|w_t), \tag{6.1}$$

where T is the number of training words, m is the size of the training context, and θ represents all variables to be optimized. The log probability $logp(w_{t+j}|w_t)$ can be trained by hierarchical softmax or negative sampling. We leverage the hierarchical softmax as the training algorithm, since it achieves better performance for infrequent words [154].

Extracting Keywords

Based on the training model, we attain the vector representation of each word in the clean reviews. For the seven user-concerned issues (*i.e.*, "crash", "battery", "memory", "network", "privacy", "spam", and "UI"), we capture 21 most related words to each issue based on the cosine similarity. Given two vectors of the app issue I and the examining word W, the cosine similarity is determined as

$$similarity = \frac{\sum_{i=1}^{n} I_i W_i}{\sqrt{\sum_{i=1}^{n} I_i^2}\sqrt{\sum_{i=1}^{n} W_i^2}}, \tag{6.2}$$

where I_i and W_i are the ith components of the vectors I and W, respectively.

By employing the cosine similarity method, we obtain 21 most similar words corresponding to each issue. Instead of regarding these words as keywords directly, we remove the stop words which occur frequently but carry fewer meanings in the reviews, *e.g.*, "a", "is", and "the". We eliminate the stop words provided by the NLTK [17] corpus. Moreover, we remove these words that appear close to the app issue in distance but not semantically related actually. To achieve this, we employ k-means algorithm to cluster all words into groups. Thus, these words in the same group are consistent semantically in theory.

Table 6.5 shows similar words and keywords of "battery". Sev-

eral similar words are removed from the keywords list because they are not in the same cluster with battery, including "cpu", "ram", "deplete", "memory", and "foreground". After getting keywords of each issue, we prioritize issues in different platforms by a keyword-based method.

Table 6.5: Similar words and keywords of "Battery"

Issue	Similar Words	Keywords
Battery	battery, drain, usage, consumption, overheat, drainer, consume, cpu, power, ram, hog, electricity, drainage, charger, batter, standby, discharge, energy, deplete, memory, foreground	battery, drain, usage, consumption, overheat, drainer, consume, ~~cpu~~, power, ~~ram~~, hog, electricity, drainage, charger, batter, standby, discharge, energy, ~~deplete~~, ~~memory~~, ~~foreground~~

Ranking Reviews

To help developers understand one specific issue deeply, we also prioritize raw user reviews regarding the issue according to their importance and usefulness for app developers. We consider one review related to the issue, if the review comprises the corresponding keywords. For all the related reviews, we rank their importance based on the lengths and ratings. Generally, reviews with lower ratings and longer lengths are preferred by developers, since they tend to express the app bugs or the necessary features. The ranking score $score(t)$ for the issue t is defined as follows.

$$score(t) = e^{-r(\frac{1}{ln(h)+1}+\frac{1}{ln(n_t)+1})}, \tag{6.3}$$

where n_t indicates the number of keywords for the issue t, r denotes the user rating, and h represents the review length. The definition ensures the ranking score to be ranged from 0 to 1. Finally, the reviews with lower ratings and longer lengths are prioritized.

6.4 Experimental Study

n this section, we first illustrate the subject datasets and performance metrics. We present the experimental results of CrossMiner on the subject datasets. To verify that CrossMiner can help developers in practice, we conduct several experiments and case studies. More specifically, we aim to answer the following three research questions:
RQ1: What are the issue distributions on different platforms based on CrossMiner?
RQ2: What is the issue-prioritization performance of CrossMiner on different app platforms?
RQ3: In addition to prioritizing app-level issues, can CrossMiner provide developers with platform-level advice?

6.4.1 Dataset

Our dataset has been collected from AppFigures [4], a website providing APIs to crawl user reviews in multiple app stores, including Google Play, App Store, Windows Store, etc. Our dataset contains 4,663,316 reviews posted by users between September, 2014 and March, 2016. 20 popular apps belonging to 8 categories are studied. Specifically, our dataset comprises 2,637,438 reviews from Google Play, 1,687,003 reviews from App Store, and 338,875 reviews from Windows Store, which are large enough for review analysis [52]. Table 6.6 lists the details of our dataset.

6.4.2 Performance Metrics

To measure the performance of the issue prioritizing results based on CrossMiner, we adopt Normalized Discounted Cumulative Gain (NDCG) in the following:

$$NDCG@k = \frac{DCG@k}{IDCG@k}, \tag{6.4}$$

Table 6.6: Review dataset of 20 subject apps.

Category	App Name	Google Play	App Store	Windows Store
Communication	LINE	102,155	104,960	9,511
	Messenger	244,516	234,400	15,801
	Skype	186,868	8,834	35,355
	Viber	161,833	109,710	21,569
	WeChat	89,205	204,922	9,508
	WhatsApp	241,792	85,117	25,130
Education	Duolingo	65,632	59,659	12,365
	TED	778	905	380
Entertainment	Netflix	97,503	45,383	28,846
	Spotify Music	178,477	249,212	33,143
	VLC	3,725	771	4,674
	YouTube	69,300	210,371	13,404
Photography	Camera360	122,350	51,777	2,319
Productivity	Evernote	65,540	30,795	2,308
Shopping	eBay	142,129	20,000	4,485
Social	Facebook	244,897	232,347	51,040
	Instagram	249,132	13,741	55,596
	Tango	122,638	200	53
	Twitter	246,546	23,200	13,218
Transportation	HERE	2,422	699	170
Total Reviews		2,637,438	1,687,003	338,875

where $NDCG@k \in [0, 1]$, with 1 representing the ideal rank order. The higher value indicates the predicted rank order is closer to the ideal rank order.

6.4.3 Keywords Generation Results

We train the word2vec model based on all the clean reviews, 3,113,111 reviews totally. As for the parameter settings, we set the word vector dimensionality as 300, the context size as 10, and the minimum word count as 80. The model training process takes several minutes to tens of minutes depending on the vocabulary size. Ultimately, we obtain the word2vec model based on our dataset. Each word in the dataset is represented by a 300-dimension vector.

Next, we extract the keywords for each issue based on the keywords generation method introduced in the step 2 in the Methodol-

ogy section. Table 6.7 depicts the relevant keywords corresponding to the seven issues.

Table 6.7: Keywords corresponding to seven issues.

Issue	Keywords
Battery	battery, drain, usage, consumption, overheat, drainer, consume, power, hog, electricity, drainage, charger, batter, standby, discharge, energy
Crash	crash, freeze, foreclose, lag, crush, stall, close, shut, laggy, glitch, hang, load, stuck, startup, buffer, open, laggs, freez, glitchy, buggy
Memory	memory, storage, space, gb, internal, gigabyte, ram, 6gb, occupy, 4gb, mb, 300mb, 8gb, 500mb, 16gb, byte, 5gb, gig, 2gb, 1gb, 1g
Network	network, connectivity, internet, consumption, wifi, connection, reception, conection, connect, signal, 4g, wi, 3g, broadband, fibre, lte, reconnecting, fi, wireless, reconnect, disconnect
Privacy	privacy, security, invade, safety, personal, policy, invasion, breach, protection, protect, private, disclosure, secure, unsafe, insecure, permission, fingerprint, encryption, violation, encrypt
Spam	spam, spammer, scammer, unsolicited, harassment, unwanted, bot, bombard, junk, scam, advertisement, popups, scraper, hacker
UI	ui, interface, design, layout, gui, ux, clunky, redesign, aesthetic, navigation, usability, desing, sleek, appearance, aesthetically, intuitive, minimalistic, ugly, slick, graphic, unintuitive

Compared to the method for selecting keywords manually, our keywords generation method has these following advantages. First, CrossMiner can automatically generate the keywords for each issue, which is more time-saving and more efficient. In contrast, manually selecting the keywords for the issues could be laborious. Second, CrossMiner can extract misspelled and made-up words that are related to the issue, which are generally ignored during the manual process. For example, as illustrated in Table 6.7, CrossMiner specifies "conection" as a keyword for the "network" issue, although it is a misspelled word of "connection". Moreover, among the keywords of the "memory" issue, made-up words (*e.g.*, "6gb", "300mb", etc.) are utilized to discuss the issue. In summary, we present an automatic and effective keywords generation method that outperforms the traditional method [97].

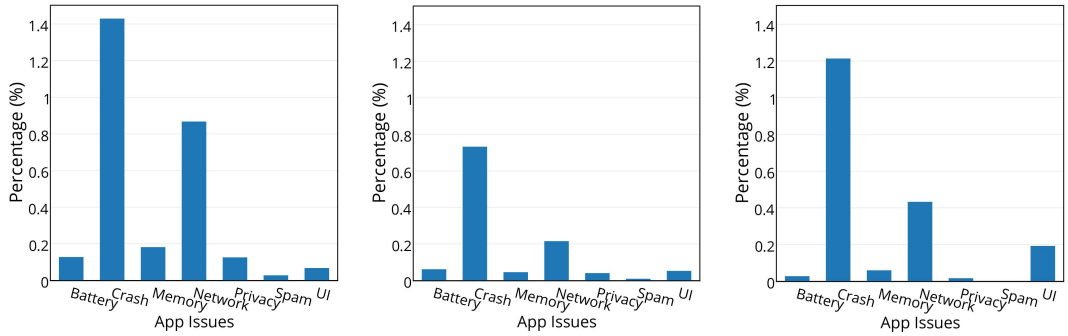
6.4.4 Answer to RQ1: Issues Distributions on Different Platforms

To answer RQ1, we conduct experiments on the 20 apps (listed in Table 6.6). In this section, we employ two apps - Spotify Music and eBay for illustration. In our dataset, Spotify Music has 178,477 reviews from Google Play, 249,212 reviews from App Store, and 33,143 reviews from Windows Store, while eBay has 142,129 reviews, 20,000 reviews, and 4,485 reviews from these three app stores, respectively. To reduce the influence of "useless" reviews, we only analyze the "informative" and "non-informative" reviews of the two apps. We determine whether a user indeed complains about a certain issue in his/her review based on two requirements: 1) The review must contain at least one of the keywords for the corresponding issue; 2) The rating of the review must be less than three stars to ensure that the reviews are expressing complaints. The experimental results of Spotify Music and eBay are discussed in the following.

Case Study on Spotify Music

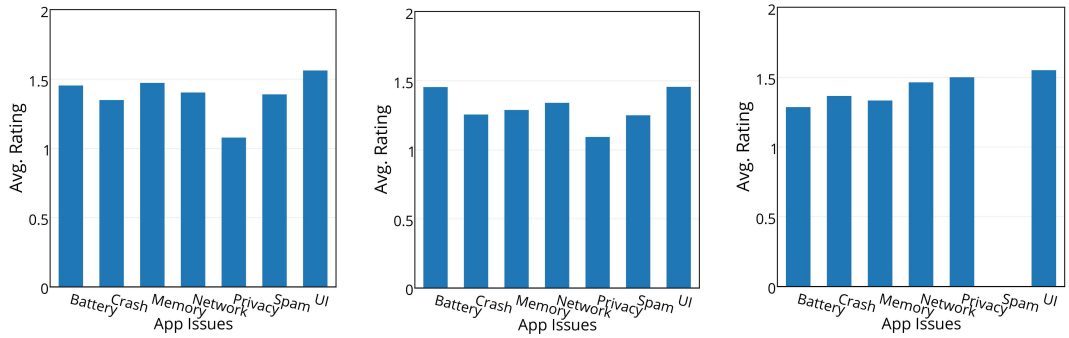
We focus on studying Spotify Music in this part. After preprocessing, Spotify Music has 154,550 clean reviews from Google Play, 217,535 clean reviews from App Store, and 25,480 clean reviews from Windows Store. The issue distributions are illustrated in Fig. 6.3 and Fig. 6.4, visualizing issue percentages and corresponding average ratings, respectively.

Results: Figure 6.3 presents the percentage distribution on the seven issues for Spotify Music. We discover that the "crash", "network" and "memory" issues are the primary concerns of Android users, accounting for 1.429%, 0.867%, and 0.181%, respectively. For the iOS users, they are more concerned about issues related to "crash" (0.732%), "network" (0.215%), and "battery" (0.061%). Among Windows Phone users, they complain more



(a) Google Play. (b) App Store. (c)Windows Store.

Figure 6.3: Percentage distribution on issues of Spotify Music.



(a) Google Play. (b) App Store. (c)Windows Store.

Figure 6.4: Rating distribution on issues of Spotify Music.

about the "crash", "network", and "UI" issues, occupying 1.213%, 0.432%, and 0.192%, respectively.

Figure 6.4 depicts the rating distribution of the seven issues. We identify that the "privacy" (1.08), "crash" (1.35), and "spam" (1.39) issues represent lower ratings than other issues in Google Play. Similarly, in App Store, the three issues also correspond to the lowest ratings, which are 1.09, 1.25, and 1.26, respectively. However, in Windows Store, the issues with the lowest ratings become related to "battery", "memory", and "crash", with average ratings at 1.29, 1.33, and 1.37, respectively.

Discussion: As Figure 6.3 illustrates, Spotify Music users of the three platforms concern more about issues relevant to "crash" and "network". Frequently crash can definitely destroy users' percep-

tions and generate unfavorable reviews. Regarding the "network" issues, since Spotify Music is a music streaming app that provides digital music service, users may feel uncomfortable or annoyed if the music downloading is too slow or consumes too much traffic. Besides the "crash" and "network" issues, for the Android platform, users also complain about "memory" (0.181%), "battery" (0.127%), "privacy" (0.125%), "UI" (0.067%), and "spam" (0.027%). With respect to the iOS platform, 0.061% users convey dissatisfaction with "battery", with other issues "UI", "memory", "privacy", and "spam" accounting for 0.052%, 0.045%, 0.04%, and 0.009%, respectively. For the Windows Phone platform, "UI" (0.192%) are more concerned by users, followed by "memory" (0.059%), "battery" (0.125%), "privacy" (0.067%), and "spam" (0.027%).

Overall, the iOS version seems to outperform the Android version and Windows Phone version. For example, the percentages of the "crash" and "network" issues for the iOS version are lower than those of the other versions. To verify the fact, we also calculate the average ratings across the three platforms. We discover that the iOS version receives the highest ratings (4.48), with the Windows Phone version (4.1) followed after. The Android version only receives 3.90 stars.

Therefore, we suggest that the Spotify Music developers should focus on testing issues related to "crash" and "network" on the three platforms, especially weighing more on the Android version. Moreover, the developers should also design comprehensive testing cases for the "memory" for the Android version, "battery" for the iOS version, and "UI" for the Windows Phone version.

To help developers gain an in-depth understanding about one specific issue, we prioritize reviews associated with the issue based on the method. Table 6.8 illustrates the top three reviews related to the "UI" issue in Google Play. As the Table shown, all the reviews complain about some aspects of "UI" (e.g., "no way to go back" in review 1, "missing basic and obvious music player features" in

review 2, and "playing view the artwork is smaller to fit in the artwork on either side" in review 3). Thus, developers can schedule the app modification based on the prioritized reviews.

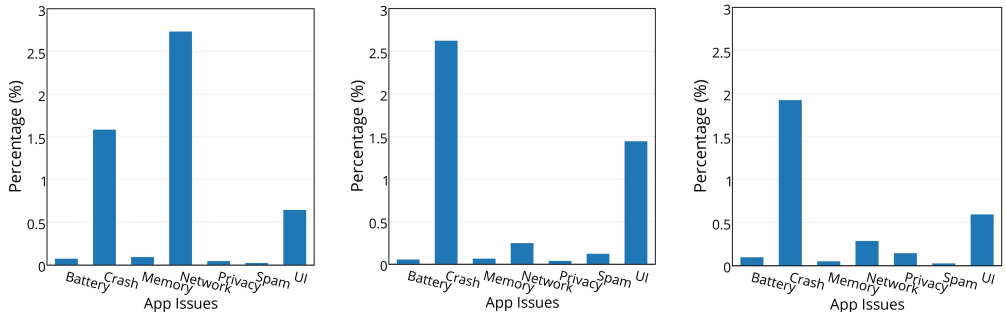
Table 6.8: Top three reviews related to "UI" of Spotify Music in Google Play

Rank	User Review	Score
1	Seriously bad user experience and interface. Once you've liked or unliked a song, there's no way to go back even if you've made a mistake. I don't know why Spotify is so popular with suck poor graphic design.	0.943
2	Clunky unintuitive interface missing basic and obvious music player features. You must get the basics right first before trying to push rubbish the user doesn't want.	0.914
3	Don't like the new design, in the now playing view the artwork is smaller to fit in the artwork on either side. I don't care what's on either end of my current playing track, or at least show it in a way that doesn't take up artwork space. The album art is always an awesome part of the music's personality so it shouldn't be minimised like this. Also the now playing bar at the bottom of the screen isn't flat looking, looks like design from windows XP. Not happy. An awesome service needs an awesome interface.	0.890
...	...	...

Case Study on eBay

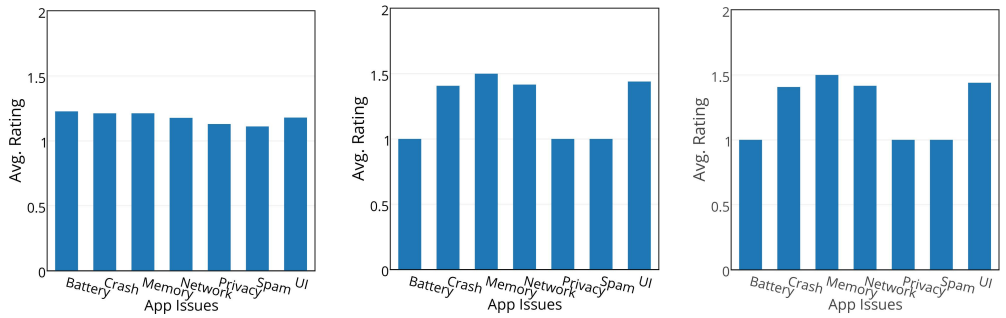
We focus on analyzing the experimental results of eBay in this part. After preprocessing, the shopping app eBay possesses 122,977 clean reviews from Google Play, 19,192 clean reviews from App Store, and 4,207 clean reviews from Windows Store. The percentage and rating distributions on the seven issues are illustrated in Figure 6.5 and Figure 6.6, respectively.

Results: Figure 6.5 describes the issue percentage distribution of eBay. As the figure illustrates, Android users are most concerned about issues related to "network", "crash", and "UI", accounting for 2.737%, 1.586%, and 0.644%, respectively. While iOS users complain more about "crash", "UI", and "network", with percentages 2.626%, 1.443%, and 0.245%, respectively. With regarding to the Window Phone platform, the users also care about the "crash"



(a) Google Play. (b) App Store. (c)Windows Store.

Figure 6.5: Percentage distribution on issues of eBay



(a) Google Play. (b) App Store. (c)Windows Store.

Figure 6.6: Rating distribution on issues of eBay

(1.925%), "UI" (0.594%), and "network" (0.285%), similar to the iOS users.

Figure 6.6 depicts the rating distribution on the seven issues. In Google Play, the "spam", "privacy", and "network" issues correspond to the lowest ratings than the others, scored at 1.11, 1.13, and 1.177, respectively. For App Store, the three issues with poorest ratings are "spam" (1.043), "battery" (1.1), and "privacy" (1.143). Regarding the Windows Store, the poorly rated issues are "battery" (1.0), "privacy" (1.0), and "spam" (1.0).

Discussion: As Figure 6.5 illustrates, users of different platforms all complain more about "crash", "network" and "UI". Differences also exist across the platforms. For example, in Google Play, 1.586% users express about the "network" issue, much more than other two platforms. Therefore, the eBay developers are suggested to focus on

testing the "network" issue for the Android version. As Figure 6.6 depicts, users tend to give poor ratings to the "privacy" and "spam" issues. In comparison, the Window Phone users are more critical, since they all rate with the lowest ratings (1.0) for these two issues.

Summary: By these two app-level case studies, we discover that users of different platforms indeed concern about different issues of an app. CrossMiner automatically prioritizes the user-concerned issues. Developers can arrange and design the testing cases for the important issues on each platform. Moreover, based on the case studies, we also identify some similarities across the platforms. For example, users generally concern more about "crash" and "network" issues.

6.4.5 Answer to RQ2: Evaluation of issue prioritization

If the prioritized issues are consistent with the practical user concerns, the performance of CrossMiner can be verified.

We employ Spotify Music for the performance verification, and the official user forums as the ground truth [25]. An issue with more user views indicates that the issue is more concerned by users. Thus, we obtain the ranks of the seven issues, with an example of Android forums illustrated in Table 6.9.

Table 6.9: Ranked issues from Android community of Spotify Music

Rank	Views	User Feedback	Issue
1	56416	No internet connection available	Network
2	32495	No SD Card storage !!	Memory
3	24797	Spotify for Android causing massive battery drain and heating of phone	Battery
4	11315	Spotify crashes on Android	Crash
5	1796	Issues with Android UI context menu touch area	UI
6	197	Intrusive or what!!!!!!	Privacy
7	80	Tired of the push notification spam!	Spam

Similarly, we capture the issue rankings from the official iOS community and Window Phone community. For the iOS version,

the ranked issues are crash(53683), memory(10797), UI(8439), battery(6174), connection(4767), spam(1903), and privacy(1558). Regarding the Windows Phone version, the issue order is crash(4097), connection(1362), battery(300), UI(282), memory(229), spam(94), and privacy(76). We then compare the prioritization results attained by CrossMiner with the ground truth for these three platforms. The NDCG@7 scores are utilized for the measurement, with results described in Table 6.10.

Table 6.10: Performance of issue prioritization

	Android	iOS	Windows Phone
NDGC@7	0.943	0.911	0.982

By examining the results, we discover that CrossMiner achieves 0.943, 0.911, and 0.982, in terms of NDGC@7 for the Android, iOS, and Windows Phone versions, respectively. The average accuracy arrives at 0.945, which indicates that CrossMiner prioritizes issues effectively and reflects the user concerns accurately.

6.4.6 Answer to RQ3: Platform-level advice to developers

In this section, we aim at exploring the platform-level issues. Figure 6.7 illustrates the issue distributions with respect to the 20 subject apps in Google Play, iOS, and Windows Phone.

Results: As Figure 6.7 depicts, the blue bar indicates the issue distributions in Google Play, with the orange bar representing the App Store and the green bar denoting the Window Store. Each bar in the graph represents the percentage of an issue. We discover that the top three issues Android users complain most about are “crash” (1.76%), “network” (0.85%), and “memory” (0.31%). Similarly, the iOS users also express more about “crash” (4.48%), “network” (1.05%), and “memory” (0.48%). For the Window Phone users, they are more concerned about “crash” (2.76%), “network” (0.66%), and “battery” (0.38%).

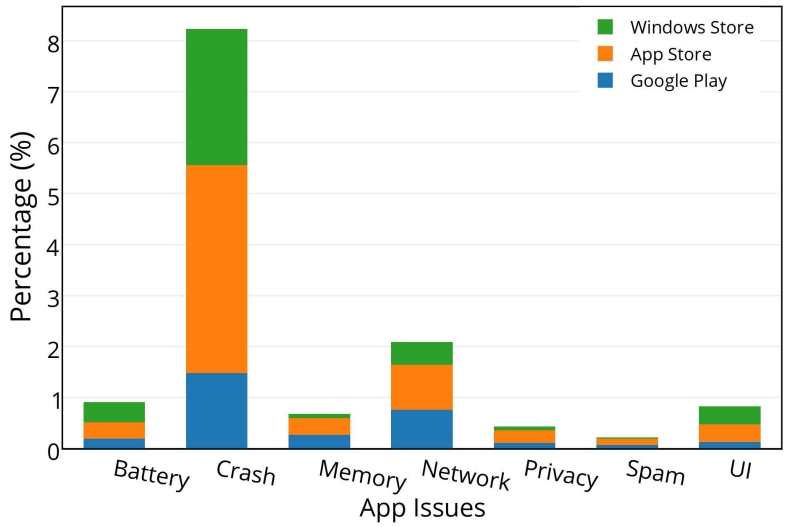


Figure 6.7: Average percentage distributions on issues for different platforms.

Discussion: We identify that users for all the platforms concern more about the "crash" and "network" issues. In Khalid *et al.*'s study [95], "app crashing" and "network problem" are ranked at the third and fourth position among all the most frequent complaints list (the top two complaints are "functional error" and "feature request", which are excluded from our study), which is compatible with our findings.

6.4.7 Parameter Study

In our framework, one key problem is to set the number of the similar words n for each issue. To obtain an optimal solution, we conduct an experimental study on the parameter settings. We first determine n for the similar words extraction, and obtain the ultimate keywords corresponding to each issue. We then define "Cover-rate" to compute the ratio of the number of keywords to the number of similar words extracted in the first step. Smaller Cover-rate indicates that the similar words comprise more unrelated words to the issue. Figure 6.8 depicts the Cover-rate along with the number of similar words n.

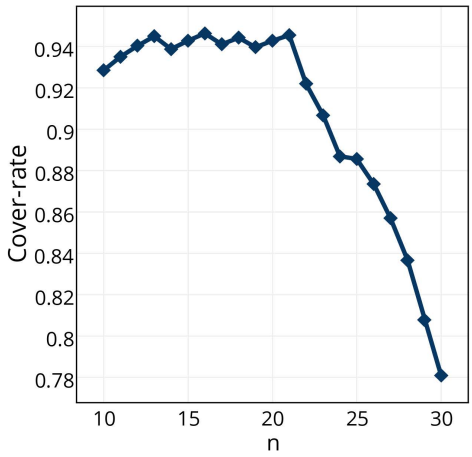


Figure 6.8: Influence of n on Cover-rate.

Figure 6.8 illustrates that fewer similar words correspond to relatively high Cover-ratio. However, some keywords will be missed if n is set too small. On the other hand, larger numbers of similar words can cover most keywords, but also carry with more unrelated words. Therefore, we set the number of the similar words n to be 21 due to the higher performance (illustrated in Figure 6.8).

6.5 Discussions

In this section, we discuss the threats to validity of our work and talk about the steps we take to mitigate these threats.

First of all, the results and conclusions of our work are based on 20 apps from Google Play, App Store, and Window Store, which is an extremely small dataset compared to all the apps in the three app stores. To mitigate the threat, we select apps that are popular among all the app stores and try to diversify their categories. This indicates that the subject apps are representative and comprehensive for the experimental study. Massive user reviews also guarantee the effectiveness of our results.

Second, apart from Android, iOS, and Windows Phone, there are other mobile operating systems, such as BlackBerry OS and

Symbian OS. It is unclear that if CrossMiner can attain good results in other platforms. Considering the three mobile operating systems have occupied 99.3% of the market, we conclude that the explored platforms are effective for verifying our proposed framework. Since we analyze the app issues from user reviews, CrossMiner can also be generalized to other platforms.

Third, we just analyze seven issues in the chapter, which may not cover all the app issues. However, since our framework can identify the keywords related to the issue effectively, other types of issues can also be analyzed similarly. This also illustrates the scalability and usability of our framework.

Finally, we are uncertain whether our discoveries can really facilitate the app testing process for app developers. Through the experiments, we determine that the same app indeed exhibits different distributions of app issues on different platforms. Developers can prioritize the testing cases accordingly, which are supposed to improve the efficiency of the testing procedure. Moreover, the prioritized issues are consistent with the issues reflected on the user forums. Therefore, we believe that our framework can facilitate the app development.

6.6 Summary

In this chapter, we propose a novel framework named CrossMiner to automatically analyze app issues from user reviews by employing a keyword-based method. We aim at discovering the differences of issue distributions on three popular app stores, *i.e.*, Google Play, App Store, and Windows Store. Based on the identified issue distributions, app developers can design and arrange the testing cases more efficiently for different platforms. To our best knowledge, CrossMiner is the first framework proposed to explore app issues on different platforms from users' perspective. The experimental study also verifies that our framework can reflect the user concerns

accurately.

□ **End of chapter.**

Chapter 7

Exploring the Effects of In-App Ads on User Experience

In-app advertising is the primary source of revenue for many mobile apps. However, ad cost, such as mobile resource occupation and customer churn, is non-negligible for app developers to ensure a good user experience and continuous profits. Previous studies mainly focus on addressing performance costs generated by ads, or resorting to surveys to collect general factors that impact users' acceptance of ads. However, users' detailed concerns about ads, and their attitude towards ads' practical performance costs (*e.g.*, memory costs) have rarely been studied. In this chapter, we prioritize concrete user concerns about in-app ads by mining massive ad-related user feedback, and explore user opinions on the performance costs of ads in practice. The main points of this chapter are as follows. (1) It proposes a measurement method for evaluating user concerns about specific app issue. (2) It analyzes ad-related user feedback and categorizes major ad issues cared by users into five types of ad issues. (3) It explores whether more performance costs of ads can generate more user concerns.

7.1 Introduction

In-app advertising is a type of advertisement (ad) within mobile applications (apps). Many organizations have successfully monetized their apps with ads and reaped huge profits. For example, the mobile ad revenue accounted for 76% of Facebook's total sales in the first quarter of 2016 [61], and increases 49% year on year to about $10.14 billion in 2017 [60]. Triggered by such visible profits, mobile advertising has experienced tremendous growth recently [32]. Many free apps, which occupy more than 68% of the over two million apps in Google Play [40], adopt in-app advertising for monetization. However, the adoption of ads has strong implications for both users and app developers. For example, almost 50% of users uninstall apps just because of "intrusive" mobile ads [33], resulting in a heavy reduction in user volume of the apps. Smaller audiences generate fewer impressions (*i.e.*, ad displaying) and clicks, thereby making ad profits harder for developers to earn. To alleviate the conflicts between users and developers, we conduct an empirical study to explore the effects of in-app ads on user experience, *i.e.*, what ad-related issues are concerned by users.

Previous research has been devoted to investigating the hidden costs of ads, *e.g.*, energy [124], traffic [129], system design [69], and other factors [74, 166], among which user surveys are commonly employed to understand users' perceptions of mobile advertising, *e.g.*, interactivity [199], perceived usefulness [168], and credibility [55]. Nevertheless, there is still a lack of systematic studies on exploring users' attitude towards in-app ads in practice, that is, whether and what users care about the displaying ads and generated performance costs during their interactions with apps. There are several challenges to this study. First, collecting massive user feedback that reflects ad-related issues by survey is intractable, since tremendous manual work will be required, *e.g.*, distributing questionnaires. Also, designing comprehensive questionnaires normally

requires a deep understanding of ad issues. Second, users' concerns about ad costs are difficult to be quantified, where user behaviors (such as app downloading and rating apps) should be well involved. Moreover, measuring the performance costs solely incurred by ads are difficult practically due to diverse usage patterns (*e.g.*, different ad viewing durations).

In this chapter, we try to overcome these challenges and focus on answering the following three questions.

a) Can ads adversely impact user behaviors towards apps? This is to investigate whether ads can reduce download times and cause lower user ratings.

b) What are the top ad issues concerned by users? We aim at prioritizing app issues produced by ads, and providing concrete suggestions on displaying ads in a user-friendly fashion.

c) How can the performance costs of ads affect user opinions? We concentrate on the performance-related cost (*i.e.*, consumption of CPU, memory, battery, and data traffic), which is intensively studied by previous work. We explore whether users express more care for more performance costs.

To answer the first question, we conduct a statistical analysis on 4,355 popular apps and another 22,327 general apps provided by PlayDrone [184]. All the apps were crawled from Google Play in May, 2018. We discover that for both popular free apps and general free apps, there exist a significant difference between apps containing ads and apps without ads regarding user ratings and the number of user ratings[1].

During answering the second question, we focus on mining ad-related reviews and manually categorize the ad issues complained by users into five types. To facilitate developers' understanding of the importance and contents of these issue types, we design an interactive and direct visualization way to display the ad issues.

[1]We use number of user ratings as an indication of download times due to the lack of such data on Google Play.

For answering the last question, we collect the usage traces of 17 volunteer users on 20 Android apps containing ads. We focus on measuring four types of performance costs, including memory consumption, CPU utilization, network usage[2], and battery drainage. The recorded usage traces are then replayed multiple times for simulating real usage scenarios and accurate cost measurement, resulting in the collection of 2,040 measurements for those apps. To quantify users' concerns about performance costs, we analyze 34,455 user reviews corresponding to the 20 subject apps from Google Play. Based on the quantified cost values and corresponding user concerns, we adopt a comprehensive correlation analysis to investigate how the performance costs of ads can affect user opinions.

Our study results in several ***interesting findings***:

- Rating alone is not enough to understand users' reaction to in-app ads.
- Irrelevant content is the most common complaint from users about ads, *e.g.*, spam ads and phishing ads.
- Some actions, such as shortening compulsory video ads, avoiding pop-up ads, and choosing appropriate sizes, can mitigate users' aversion to ads.
- Users are more concerned about the battery costs of ads, and tend to be insensitive to other performance cost types (*i.e.*, CPU, memory, and data traffic).

The key ***contributions*** of this chapter are as follows.

(a) We propose a novel method for automatically prioritizing users' concerns towards ads from user reviews, by involving user sentiment and feedback volume. We visualize user-concerned ad

[2]Network usage is included here since it can impact mobile performance [182].

issues in an interactive and directly-perceived form for facilitating developers' observation.

(b) We provide detailed findings and insights captured from extensive analysis on user feedback, which would be referred by app developers when designing ads.

(c) This is the first work to explore the correlations between user concerns and the practical performance costs of ads, from which we deduce whether these costs can influence user opinions.

(d) We conduct survey on app developers to gain deep knowledge of users' concerns about in-app ads and whether our findings are beneficial for developers, respectively.

(e) The source code for cost measurement and user review analysis used in our study has been made publicly available[3] for allowing developers and researchers to use the code for their own purposes.

7.2 Methodology

Figure 7.1 presents an overview of the methodology used during our exploration about in-app ads. First, we conduct statistical analysis on 4,355 popular apps and 22,327 general apps crawled from Google Play. The two categories of collections have 1,084 overlapping apps. We analyze whether in-app ads can influence user ratings and number of user ratings for free apps. We will elaborate the datasets and our findings in the experimental section.

Then to identify the top ad issues concerned by users, we extract those reviews describing mobile ads. To this end, we retrieve the terms (including phrases and single words) that frequently co-occur with "ads". Based on sentiment analysis, we propose an approach for quantifying user concerns about each ad issue. The details will be illustrated in Section 7.2.1, with the visualization manner of the captured ad issues depicted in Section 7.2.1.

[3]https://remine-lab.github.io/adbetter.html

Finally, we analyze users' attitude towards the performance costs of ads, focusing on the consumption of CPU, memory, battery, and data traffic. Similar to [74], the measurement of performance costs in our work is also implemented on instrumented mobile phones. We start by installing the subject apps into the instrumented phones. The measuring strategies of different types of performance costs are explained in Section 7.2.2. The usage patterns of users using the subject apps are collected for practical cost measurement (will be introduced in Section 7.3.3). For gauging user concerns of these performance costs, we employ the similar approach for quantifying users' attitudes towards ad issues (Section 7.2.3). Based on the measured user concerns and performance costs in practice, we conduct correlation analysis to determine whether users care about performance costs (Section 7.2.4).

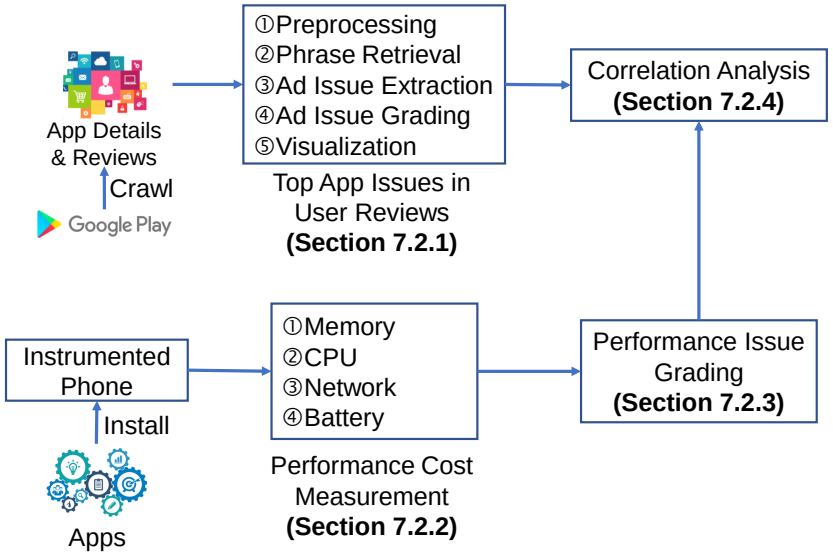


Figure 7.1: Workflow of our study.

7.2.1 Top Ad Issues in User Reviews

In this section, we explore how to capture top ad issues and quantify users' complaints about these issues.

The pipeline of our review analysis is depicted in Figure 7.1. First, we preprocess raw app reviews to obtain structured reviews (Section 7.2.1). We extract the reviews that are explicitly related to ads, *i.e.*, reviews containing words such as "ad", "ads", or "advert*" (with regular expression). These reviews are referred to as ***ad-related reviews*** throughout this chapter. Then we retrieve key phrases (*i.e.*, more than one word) that are commonly used in user reviews as phrase candidates (Section 7.2.1). We identify the terms (including phrases and single words) that are commonly adopted to describe mobile ads in an unsupervised way. The identified terms are the descriptions of ad issues. We manually categorize the ad issues into five types, and group review sentences according to the issue types (Section 7.2.1). Then, we present our methods for measuring users' concerns about different issue types based on sentiment analysis and their proportions among user feedback (Section 7.2.1). Finally, to assist developers in understanding these ad issues, we visualize their importance in an intuitive way (Section 7.2.1).

Preprocessing

App reviews are usually short in length and contain many casual words. To facilitate subsequent analysis, we eliminate the noisy characters in this step. We first convert all words into lowercase, and remove all non-English characters and non-alpha-numeric symbols. We retain the punctuations to ensure semantic integrity. Then, we reduce the words to their root forms by lemmatization [17], *e.g.*, "was" to "be". Finally, we keep reviews with the number of words larger than three. We do not remove stop words [136] here for phrase retrieval in the next step. Since app reviews contain growing compound words (*e.g.*, redownload), new words (*e.g.*, galaxys8), and misspelled words (*e.g.*, updte [update]), we do not involve the preprocessing methods in [186] where the custom dictionary may introduce information loss (*e.g.*, over correction) for our situation.

Phrase Retrieval

Phrase retrieval aims to identify the key terms (particularly those with multiple words) that are commonly used by users to voice their experience. The phrases are extracted here since one single word may be ambiguous in its semantic meanings without the context information.

However, given that user reviews are casually written, extracting the meaningful phrases poses a challenge. In this chapter, we adopt the typical Point-wise Mutual Information (PMI) method [104]. The PMI method measures the co-occurrence probabilities of two words, and thereby can eliminate terms which are rarely used. The phrases we retrieve contain 2-gram terms (*i.e.*, two consecutive words) and 3-gram terms (*i.e.*, three consecutive words). Since phrases with more than three words rarely exist in the review collection, they are not extracted here. Equation (7.1) defines the PMI between two words w_1 and w_2:

$$PMI(w_1, w_2) = \log \frac{Pr(w_1\ w_2)}{Pr(w_1)Pr(w_2)}, \tag{7.1}$$

where $Pr(w_1\ w_2)$ and $Pr(w_i)$ denote the occurrence probabilities of the phrase $(w_1\ w_2)$ and the single word w_i, respectively. The terms with higher PMIs indicate that they appear together more frequently and tend to be semantically meaningful. The PMI thresholds are experimentally set. Based on the PMI results, we also ensure that at least one noun is included in each phrase via the Part-Of-Speech tagging method [171].

Ad Issue Extraction

Automatically clustering ad issues from ad-related reviews is difficult. Because it would not be easy to predefine the number of clusters and explain the topics of each cluster for unsupervised clustering methods, such as k-means [186] and topic modeling [52]

methods [35, 54]. We propose to employ a simple method based on the neural network model *word2vec* [121] to discover similar words and phrases. The model represents input terms as low-dimensional (*e.g.*, 200) dense vectors by capturing their semantic relations. The model is effective in retrieving similar words, where the semantic similarity between two terms can be computed as the cosine distance of their vector representations. We retrieve k (*e.g.*, 50) terms most similar to the words "ad" or "ads" using the model, *e.g.*, "banner ad" and " popups". Then we manually trim the noise words and phrases (*e.g.*, "gimmick" and "commercial"), and cluster the remained terms (called **ad-related terms**) into w types, *i.e.*, $\{I_1, I_2, ..., I_i, ..., I_w\}$. The box below shows the terms for the ad size type. The review sentences containing any terms in the dictionary I_i will be grouped into the issue type I_i.

Ad Size: *fullscreen ad, banner ads, full screen ad, banner ad.*

Ad Issue Grading

In this section, we explain the novel method we propose for measuring users' concerns about specific ad issues, *e.g.*, the ad size. Similar to previous work [52, 190], we assume that issues complained about in more reviews and yielding poorer ratings indicate higher concern levels among users. The time information (used by [52]) of the issues is not considered here, since we do not care about whether one issue is fresher than others.

Sentiment Score: We have obtained the reviews related to each type of ad issues discovered in Section 7.2.1. Since one piece of review may describe several app aspects with diverse sentiment, *e.g.*, one user of *br.com.ctncardoso.ctncar* commented that "*Good design and easy use. Free version includes too much advertising. You need to setup regular payment to get pro version which is ridiculous.*", the users' attitude towards in-app ads should be assessed per sentence. In our study, we chunk the reviews into sentences by utilizing

NLTK's punkt tokenizer [22]. Inspired by Guzman and Maalej's work [76], we employ SentiStrengh [175], a lexical sentiment extraction tool specialized in dealing with short, low-quality texts, for the sentiment analysis. Although Gu and Kim's study [72] improve the performance of aspect sentiment analysis in [76], we choose to use the approach proposed by Guzman and Maalej here for the efficient implementation.

SentiStrength assigns a positive and negative value to each review sentence, with positive scores in the range of [+1, +5], where +5 denotes an extremely positive sentiment and +1 denotes the absence of sentiment. Similarly, negative sentiments with the range [-5, -1], where -5 denotes an extremely negative sentiment and -1 indicates the absence of any negative sentiment. Table 7.1 displays examples of SentiStrength scores for review sentences. If the negative score multiplied by 1.5 is larger than the positive score, the sentence is assigned the negative score. Otherwise, the final sentiment score is defined as the positive score. As explained in [76], multiplying the negative scores by 1.5 is considered due to users tend to write positive reviews [90]. The sentiment score of issue type I_i in the m-th app is the average sentiment scores of all the contained review sentences, indicated as R_m^i, where m indicates the m-th app.

Frequency Score: The number of the reviews for issue type I_i in the m-th app can then be calculated, denoted as N_m^i.

Concern Score: The final concern score is defined in Equation (7.2), for measuring the user concern U_m^i by combining the two factors.

$$U_m^i = -\log f(R_m^i) \times P_m^i, \tag{7.2}$$

where $P_m^i = N_m^i/N_i$, representing the percentage of the I_i-related reviews in the m-th app reviews. The function $f(R_m^i)$ is to confine the rating R_m^i to be in the range $(0, 1)$, which is empirically defined as the soft division function, *i.e.*, $(R_m^i - 0.9)/5$. Equation (7.2) shows that issues with lower user ratings and larger review percentages will

Table 7.1: Example of SentiStrength scores and defined sentiment scores for example review sentences.

Review Sentence	SentiStrength Score	Defined Sentiment Score
Great but why make a browser if you don't have the resource to keep it up to date? the last update be april, it needs update.	[3,-1]	3
Would be 5 stars if i could pay and remove all the ads.	[1,-1]	-1
I like what it does but the additional stuff is annoying, eg loud video advert is disturbing.	[2,-3]	-3

be given higher user concern values, which is consistent with our assumption.

Ad Issue Visualization

To better illustrate the ad issues, we visualize them into a bubble graph. Each issue corresponds to a bubble, and the distance between the two bubble centers represent the cosine distance of their vector representations under *word2vec*. We use bubble sizes to denote the quantified user concerns regarding the issues, as shown in Figure 7.3. Issues with similar colors belong to the same issue types. Larger bubbles indicate that the represented issues are more cared about by users. The implementation of such visualization is based on gensim [153] and Google Chart [68]. To get the details of each issue, one can click the bubble to view the corresponding concern degree and other information.

7.2.2 Performance Cost Measurement

In this part, we elaborate on the measurement methods for each type of performance costs, including the consumption of memory, CPU, network, and battery. For precise estimation, we measure those costs

of both with-ads and no-ads versions of each app, denoted as C_m^g and $\widetilde{C}_m^g$ respectively, where $g \in \{memory, CPU, network, battery\}$, and m is the app id for the subject apps. The costs of ads A_m^g are then calculated by Equation (7.3):

$$A_m^g = C_m^g - \widetilde{C}_m^g. \tag{7.3}$$

Two mobile devices are used for measuring the performance costs of ads, including a Nexus 5 smartphone with a rooted Android 5.0.1 operating system, and a Nexus 6P smartphone with a rooted Android 6.0.1 operating system. Both phones are instrumented with `Xposed`. Below we introduce how we generate no-ads versions of subject apps, how we collect actual usage traces for cost measurement, and also the measurement methods.

Generation of No-Ads Versions

The no-ads versions of the subject apps are generated for measuring the performance costs of ads, as shown in Equation (7.3). For no-ads versions, we activate the module `AdBlocker Reborn` [34] provided by `Xposed` [194]. The module removes in-app ads according to a pre-loaded list of ad-related activities (*e.g.*, "com.google.android .gms.ads.AdView" for AdMob) and layouts (*e.g.*, "android.webkit.-WebView"). To check whether the ads have been successfully removed, we ensure that there are no ad-related requests from the no-ads versions with `tcpdump` [174]. Also, we ensure that the app structures, including the layout and functionality, are same as the original apps. Note that the module `AdBlocker Reborn` introduces negligible influences to the mobile performance, meaning that we can calculate the ad costs directly by subtracting the costs of the no-ads versions from those of the with-ads versions.

Collection of Usage Traces

To measure the practical ad costs, we collect the usage traces of subjects apps from volunteer users and on the instrumented phones without the `AdBlocker Reborn` module activated. We invite the users to exercise the functionalities of the subjects according to their own usage habits. Also, only one subject app is interacted with at one time to avoid interference from other apps. Each app is closed after use. For each volunteer, we introduce our rule (*e.g.*, one app at one time) and the functionalities of each app at the beginning. We have no limitations to their interaction duration with apps. The usage traces are collected in a safe and casual way using the `getevent` tool [66].

Measurement of Costs

We measure the performance costs generated by each usage trace as below.

Memory Consumption: We measure the memory consumption by employing a standard tool `top` [39] in Android, and with the metric Resident Set Size (RSS). RSS indicates the portion of memory occupied by a process in the main memory (RAM). The process name of each subject app can be obtained via `appt` [38], an Android asset package tool. We run the `top` tool in one second interval, and compute the average value for analysis.

CPU Utilization: Ad display involves additional loading and processing of images or videos, which may incur high CPU usage. High CPU usage may cause performance lags. We again leverage the tool `top` to monitor the CPU occupancy rate of a process per second. The average CPU utilization is calculated for our analysis.

Network Usage: We measure this type of cost based on the total bytes transmitted. The metric is estimated by adopting a common tool `tcpdump`. It starts when the app is launched, and captures all the data transmitted during the app runtime.

Battery Drainage: Since using external devices, such as Monsoon Power Monitor [125], cannot measure the power consumed by an app, we determine to adopt an available measurement framework `AppScope` [198]. `AppScope` shows an error rate lower than 7.5% in estimating battery costs. The framework provides the battery usage measurement based on five components, including CPU, LCD, WiFi, cellular and GPS. To simplify our measurement process, we switch off GPS and cellular signals and consistently set the LCD settings. We, therefore, ignore these components and focus only on measuring the power consumption of CPU and WiFi, *i.e.*, P^{CPU} and P^{WiFi}. The total battery consumption P is calculated via Equation (7.4).

$$P = P^{WiFi} + P^{CPU}, \tag{7.4}$$

which combines the battery drainage of both WiFi and CPU, and P^{WiFi} and P^{CPU} are calculated via `AppScope`.

The measurement methods we used for memory, CPU and network were also employed by [74]. Based on the collected usage traces, we leverage the tool `RERAN` [67] to replay these events, during which we record the performance-related data. To mitigate background noise, we restore the system environment to its original state before each version execution. Then we install the app and start its execution. When a subject app is launched, the tools `tcpdump` and `top` are started to capture the transmitted data traffic and memory/CPU consumption, respectively. We also monitor the app execution to ensure that they are consistent with the records. We run each of the usage traces three times for both the with-ads and no-ads versions in order to measure the costs and take the average to minimize measuring errors.

7.2.3 Performance Issue Quantification

In this part, we illustrate the procedures for quantifying user concerns about the studied performance issues. We measure users' attitudes towards the performance costs of both the with-ads apps and in-app ads, denoted as U_m^g and $\mathcal{A}_m^g$ respectively, where $g \in \{memory, CPU, network, battery\}$, and m is the app id for the subject apps.

Cost-Related Dictionary Construction

Similar to ad issue extraction explained in Section 7.2.1, we employ *word2vec* model to capture top k terms (including phrases and single words) that are semantically close to the target cost type. For example, to build the ***battery-related dictionary***, we collect the terms most relevant to the word "battery". We then manually remove ambiguous and noise ones from the collected terms, such as the terms "data volume" and "data plan" in the box below this paragraph. The remaining terms constitute the battery-related dictionary. We note that misspelled words (*e.g.*, "batery") can also be captured through *word2vec*. The ultimate dictionaries are utilized to group review sentences into different performance costs in the next step.

Battery-related dictionary: *battery life, ~~data volume~~, batery, battery power, ~~data plan~~, battery juice, ...*

Sentence Grouping

Let s be one review sentence and g be one cost type, where $g \in \{memory, CPU,$
$network, battery\}$. The method for grouping sentences is illustrated in Equation (7.5).

$$P_{s,g} = \frac{|Dict_g \bigcap Sent_s|}{|Sent_s|}, \tag{7.5}$$

where $Sent_s$ and $Dict_g$ denote the terms in sentence s and in the dictionary related to the cost type g, respectively. $|\cdot|$ is the number of terms in the collection $\cdot$. Similar to Di Sorbo *et al.* [167], $P_{s,g}$ is greater than 0.05 for avoiding mis-grouping. Also, one sentence can be assigned to one or more performance cost types.

Performance Issue Grading

In order to grade performance issues, we adopt a method similar to the one we proposed for ad issue grading in Section 7.2.1. The number and average sentiment score of the reviews for cost type g in the m-th app can be calculated, denoted as N_m^g and R_m^g respectively. Equation (7.6) measures the user concern U_m^g by combining the two factors.

$$U_m^g = -\log f(R_m^g) \times P_m^g, \tag{7.6}$$

where $P_m^g = N_m^g / N_m$, representing the percentage of the g-related reviews in the m-th app reviews. The function $f(R_m^g)$ is to limit the rating R_m^g to be within the range $(0, 1)$, which is empirically defined as the soft division function, *i.e.*, $(R_m^g - 0.9)/5$.

To capture users' concerns about the performance costs of ads, we group the ad-related reviews to different cost types via the method in Section 7.2.3. However, directly employing the grouping method may not correctly distinguish whether costs are caused by ads or not. For example, we cluster the review in the box below as one memory-related review via the method (due to the memory-related terms "memory" and "storage"), but for the user, the memory issue is not related to ads. Therefore, such reviews cannot be grouped into the memory costs of ads. To identify whether the performance costs are caused by ads, we analyze the reviews from the sentence level and focus on the ad-related sentences. In the review example, the second sentence is ad-related (due to the word "ads") and has no terms related to performance costs, and thereby we

will not group the review into any cost types of ads. Based on the grouped ad-related reviews, the calculation of user concerns about the performance costs of ads $\mathcal{A}_m^g$ is similar to Equation (7.2).

$$\mathcal{A}_m^g = -\log f(\mathcal{R}_m^g) \times \mathcal{P}_m^g, \tag{7.7}$$

where $\mathcal{R}_m^g$ and $\mathcal{P}_m^g$ denote the average rating and proportion of the ad-related reviews under the cost type g respectively. U_m^g and $\mathcal{A}_m^g$ measure the user concerns about apps and in-app ads respectively.

Memory-related review: *"I'm not the only one who has problems with memory storage. Also, less ads if possible."*

For each performance cost type g, we implement the correlation analysis on the two observations $\{A_1^g, A_2^g, ..., A_n^g\}$ and $\{\mathcal{A}_1^g, \mathcal{A}_2^g, ..., \mathcal{A}_n^g\}$.

7.2.4 Correlation Analysis

In this part, we detect how the performance costs of ads can affect user opinions by investigating the correlations between the measured performance costs $\{A_1^g, A_2^g, ..., A_n^g\}$ and corresponding user concerns $\{\mathcal{A}_1^g, \mathcal{A}_2^g, ..., \mathcal{A}_n^g\}$ on the n subject apps, where $g \in \{memory, CPU, network, battery\}$. To comprehensively analyze the correlations, we employ the Pearson correlation coefficient (PCC) [144] for detecting a linear correlation between two variables, and Spearman rank correlation (SRC) [169] for measuring their monotonic relationship in ranking. The results of PCC r_p and SRC r_s range from -1 to 1, with higher absolute values signifying stronger correlations. The signs of r_p and r_s indicate a positive correlation or negative correlation. Larger absolute values of r_p and r_s signifies that the relationship is much stronger (*e.g.*, $r \in [0.6, 0.8)$ means a strong relationship, and within the scope of $r \in [0.4, 0.6)$ implies a moderate relationship) [59]. The results with p-value≤ 0.05 indicate that the correlations can be considered statistically significant [138]. For example, for PCC, p-value≤ 0.05 means that the two variables have a strongly linear relationship.

7.3 Experiments

In this section, we describe the extensive experiments we conduct to answer the three research questions introduced in the Introduction part, including whether ads can impact user ratings (see 7.3.1), the top ad issues concerned by users (see 7.3.2), and user opinions about the performance costs of ads (see 7.3.3).

7.3.1 RQ1: Do in-app ads negatively impact user ratings?

To answer this question we analyze if the rating of apps containing ads is generally lower than the rating of those apps containing no ads. If so, this may provide developers with an initial alarming indication of the impact of ads in their apps. Otherwise more sophisticated approaches are needed to sought users' reaction to in-app ads. In the following subsections we first describe the experimental data we used and then illustrate the analysis of our results.

Dataset of popular apps and general apps

In order to investigate the relationship between in-app ads and user ratings/number of users we collect information about two sets of apps (popular apps and general apps) available from the Google Play Store, thus mitigating the app sampling problem [115]. For collecting information on popular apps, we took a snapshot of the 4,355 popular apps available in the 45 categories of Google Play on May 24th, 2018. For collecting general apps, we utilize the app ids provided by PlayDrone [184], a public collection of Android apps and metadata, as input of our customized crawler. PlayDrone offers 1,402,894 app ids in total. Due to existing deprecated apps and time limit, we captured the updated details of 22,327 general apps across 48 categories in May 2018. Table 7.2 and Table 7.3 list the number of apps in each category of popular apps and general

Table 7.2: Distributions of Experimental Popular Apps on Google Play.

Category	#Apps	Category	#Apps
Productivity	45	Arcade	36
Entertainment	180	Personalization	233
Food & Drink	111	Health & Fitness	148
News & Magazines	15	Beauty	63
Libraries & Demo	100	Educational	8
Music & Audio	157	Dating	553
Racing	10	Events	10
Travel & Local	53	Photography	224
Parenting	90	Adventure	46
Maps & Navigation	143	Books & Reference	93
Casual	30	Tools	208
House & Home	75	Weather	16
Video Players & Editors	152	Puzzle	20
Communication	23	Business	146
Education	27	Finance	7
Strategy	2	Shopping	24
Action	55	Social	128
Casino	2	Medical	1,051
Comics	97	Art & Design	47
Lifestyle	130	Sports	9
Word	2	Card	2
Auto & Vehicles	111	Simulation	10
Puzzle	20	Music	6

apps, respectively. The two sets have 1,084 overlapping apps. The information collected include app name, user rating, category, price, number of user ratings, containing ads or not, and whether offering in-app purchase. No information about developers and users were collected.

Do in-app ads impact user ratings?

As users may express different feelings for ads in free apps and those in paid apps, we analyze the relationships between ads and user ratings for these two types of apps separately. Table 7.4 depicts the distributions of experimental popular apps. We can observe that apps with ads account for the largest proportions among both popular and general apps, which reflect the importance of our explorations on in-app ads. Due to the limited the number

Table 7.3: Distributions of Experimental General Apps on Google Play.

Category	#Apps	Category	#Apps
Educational	334	Productivity	691
News & Magazines	481	Entertainment	1,327
Libraries & Demo	50	Sports	749
Food & Drink	78	Music	55
Board	175	Word	203
Trivia	155	Racing	646
Events	4	Art & Design	17
Dating	17	Shopping	353
Arcade	970	Business	274
Photography	677	Parenting	29
Adventure	205	Maps & Navigation	286
Music & Audio	742	Card	319
Auto & Vehicles	36	Lifestyle	798
Travel & Local	469	Role Playing	184
Comics	73	Medical	139
Casino	207	Simulation	437
Social	356	Action	489
Health & Fitness	423	Strategy	237
Finance	494	House & Home	42
Beauty	13	Communication	574
Puzzle	1,155	Video Players & Editors	328
Weather	198	Personalization	1,418
Education	916	Tools	1,858
Casual	1,852	Books & Reference	794

of paid apps collected, we focus our subsequent analysis on free apps. Specifically, we use the Wilcoxon rank-sum test [193] to compare the distributions of user ratings of different app groups, *i.e.*, free apps with ads (Group A) and free apps without ads (Group B). The Wilcoxon rank-sum test is an unpaired, non-parametric statistical test checking the null hypothesis "two input distributions are identical". If the p-value computed by Wilcoxon rank-sum test is smaller than 0.05, we reject the null hypothesis and conclude that the two input distributions are significantly different. On the other hand, if the p-value is larger than 0.05, we cannot claim any difference between the two input distributions. We use an unpaired test because an app cannot have a version with ads and a version without ads on Google Play meanwhile.

In addition, we consider a non-parametric effect size measure, namely the Vargha and Delaney's A_{12} statistic [183], to assess whether the effect size is worthy of interest. The A_{12} measure is agnostic to the underlying distribution of the data, and thereby applicable for our situation. We use the following thresholds for interpreting, according to [183]:

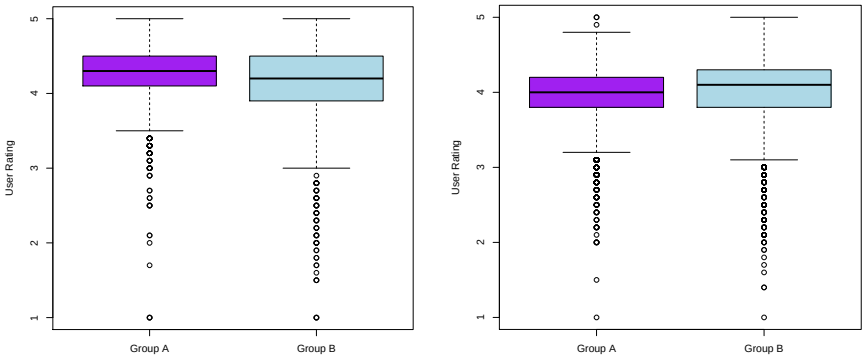
$$Effect\ Size = \begin{cases} negligible, & \text{if } A_{12} \leq 0.56. \\ small, & \text{if } 0.56 < A_{12} \leq 0.64. \\ medium, & \text{if } 0.64 < A_{12} \leq 0.71. \\ large, & \text{if } 0.71 < A_{12}. \end{cases}$$

Figure 7.2 (a) illustrates the user rating distributions in different groups of popular apps. The Wilcoxon rank-sum test shows that the two distributions are significantly different (with p-value=1.52e-21$\ll$0.05), however with a small A_{12} value at 0.58. Figure 7.2 (b) displays the user rating distributions in different groups of general apps. The Wilcoxon rank-sum test shows that the two distributions are significantly different, but with a negligible effect size ($A_{12} = 0.47$).

These results suggest that free popular apps with ads present slightly better user ratings than those apps without ads, but the difference is small. On the other hand, free general apps with ads obtain slightly lower user ratings than those apps without ads, but the difference is negligible.

Based on these results, we conjecture that free popular apps containing ads tend to be designed with more friendly app functionalities for attracting users to use, and thereby receive higher user ratings. On the other hand for some general apps, developers may not devote lots of effort into creating them, and inserting annoying ads into apps, and therefore these apps get slightly more negative ratings. Our results are in line with a previous study by [158] that explored whether the number of ad libraries impacts app ratings and find no correlations between the number of ad libraries in a given app is related to its rating in the app store.

Our analysis has revealed that in-app ads do not always lead to poor user ratings yet previous work show that they affect users reactions [95]. This simple sanity check has confirmed that apps' rating alone might not be enough to understand users' reaction to in-app ads. Thus, it is important to find other ways to explore users' opinion about in-app ads in order to help developers understand ads' impact, design user-friendly ads and ensure a good user experience.



(a) Popular apps (b) General apps

Figure 7.2: Rating distributions in different app groups. Group A and Group B indicates free apps with ads and free apps without ads, respectively.

Table 7.4: Distributions of Experimental Popular and General Apps on Google Play.

Group	Popular Apps		General Apps	
	With Ads	Without Ads	With Ads	Without Ads
Free Apps	2,172	1,688	16,214	5,840
Paid Apps	10	485	31	242

7.3.2 RQ2: What are the top ad issues concerned by users?

In this section, we aim to detect what users really care about with respect to in-app ads. To answer this question, we focus on 19,579 ad-related reviews mined from our review collection by utilizing the method in Section 7.2.1. The review collection contains 2,637,438 Google Play reviews used in [112]. We then identify 22 ad-related terms based on the method proposed in Section 7.2.1. These terms are descriptions of ad-related issues. We evaluate users' concerns about each issue based on the proposed grading strategy in Section 7.2.1, and visualize them in Figure 7.3. We manually classify the ad issues into five issue types according to their semantic meanings and illustrate different types with different colors. Larger bubbles indicate that the corresponding issues are of more concern to users.

In this part, we elaborate on the five types of ad issues by combining user reviews with the feedback from volunteer users, as some volunteers also complain to us about the badly-designed in-app ads during their interactions with subject apps.

Ad Content

Based on the review analysis, we discover that ad contents receive the highest concerns (64.0%) from users (shown in Figure 7.3), *e.g.*, reviews with "spam" occupy 59.9% of the ad review collection. This indicates that ads with junk or unwanted information are likely to receive unfavorable user feedback. For example, users complain about "Annoy ad and notification spam." for the app

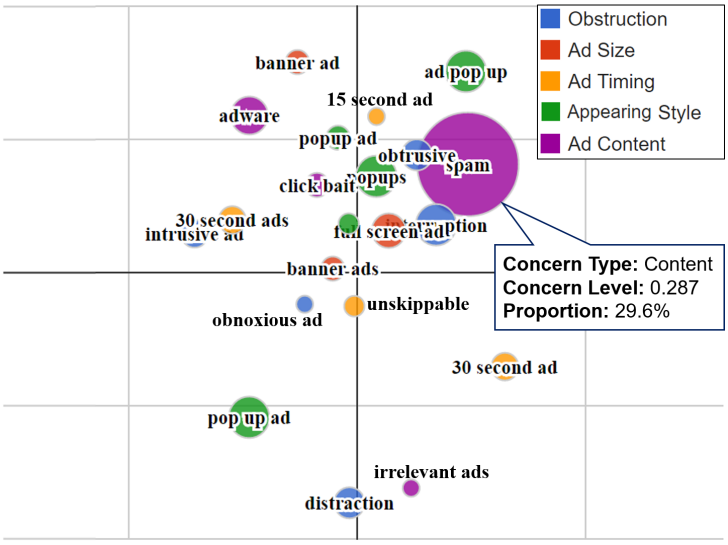


Figure 7.3: Visualization of ad issues.

com.myfitnesspal.android, and "Spam bogus ad is annoying." for the app *com.droid27.transparentclockweather*. When users are frequently shown irrelevant ads, they tend to uninstall the apps. During our experiments, two volunteers clearly describe the contents delivered by in-app ads as not what they want, saying they would uninstall such apps immediately. Also, according to the study [130], ad displaying without considering the content of the page or user preference is one key reason for reduced users' dwell time on ads, and declining developers' ad benefits. For example, displaying gambling ads in a bible app is not uncommon for users. Thus, developers should choose ad SDKs with good performance in recommending relevant ads.

Insight 1: *App developers should choose ad SDKs with good performance in recommending relevant ads to users.*

Appearance Style

The appearance styles of ads, such as popping up, significantly influence users' acceptance of ads. Annoyed users may post negative reviews with terms, such as "ad pop up", "pop up ad", and "popups",

which account for 14.7% of all ad issues. For example, one user complains that "The same reviews repeat all the time, especially the pop-up ad is very annoying." (bbc.mobile.news.ww). Such ads can interfere with the user experience of the apps, since pop-up ads may interrupt users' interactions with apps and generate inadvertent clicks on the ads. This is reflected in that "click bait", whose main purpose is to attract users to click and visit particular web pages, is close to the issue of "popups" in Figure 7.3. During the experiments, four participants state that the pop-up ads usually appear near the buttons and are prone to being clicked (*e.g.*, for the app *com.avg.cleaner*). One person also states that it is irritating when ads with sizes larger than banners pop up in the center of the screen (*e.g.*, com.wunderground.android.weather). Thus, developers should avoid embedding pop-up ads into apps.

Insight 2: *App developers should avoid pop-up ads in the apps and place the ads away from the gadgets (e.g., buttons) that are inclined to be clicked by users.*

Ad Size

Generally, developers are required to define the ad display sizes, mainly including full-screen ads (*i.e.*, interstitials) and banner ads (*i.e.*, banners). As Figure 7.3 depicts, the issues related to ad sizes, represented as red bubbles, occupy a certain proportion of all the ad-related issues (3.7%). The negative reviews about ad sizes are usually described with terms like "banner ad" and "full screen ad". We also discover that the "full screen ad" is more concerned by users than "banner ads" (0.7% more), which implies that full screen ads are more disruptive to users. During the experiments, two respondents also complained that the interstitials with unobservable closing buttons are less enjoyed by users. Such annoying settings may provoke users to uninstall the apps. Hence, developers should choose ads with appropriate sizes and visible closing symbols.

Insight 3: *App developers should avoid full-screen ads during the app usage, and provide distinguishing closing symbols in both banners and interstitials.*

Ad Timing

Ad timing refers to a particular ad display time that users cannot skip. This kind of ad issues mainly occurs with video ads and occupies 3.0% of all ad issues. Users are compelled to view ads at the beginning or end of an activity without any skipping choices, which may cause discontents in users. Annoyed users may express this with "unskippable", "30 second ads", "15 second ad" and other terms with time settings in reviews. For example, one review mentions "A 15 seconds ad play before any video content which is really annoying." (bbc.mobile.news.ww). During the experiments, two participants complain that the 15-second ads before watching news are very disturbing. Moreover, these ads are not skippable, reflected in the reviews like "The ad is most annoying as they are repetitive and there is no way to skip them." (bbc.mobile.news.ww) and "Shouldn't force to watch the same 30 seconds ad over and over without option to skip ads." (con.foxnews.android). Thus, developers should display short video ads in apps and provide users with skip permissions.

Insight 4: *App developers should shorten the compulsory video ads and provide clear skipping or closing options for users.*

Obstruction

The previous work [181] shows that almost 50% of users uninstall apps just because of intrusive advertising. In Figure 7.3, the characteristics of such obstruction are reflected in terms such as "intrusive ad", "obtrusive", and "obnoxious ad". This type of ad issues occupies a large proportion (14.7%) of all the ad issues. As Figure 7.3 shows, terms related to "obstruction" are close to

those related to the appearance style of ads (“popups”), ad contents (“spam”), ad timing (“30 second ads”), and ad sizes (“full screen ad”). This indicates that all the ad issues above are correlated with the “obstruction” feature of ads. Developers should therefore pay much attention to ad design and provide user-friendly mobile advertising that does not interfere with the user experience.

7.3.3 RQ3: How can the performance costs of ads affect user opinions?

To answer this question, we conduct a correlation analysis on the measured performance costs of ads and corresponding user concerns. In this section, we first detail the subject apps and collected usage tracts. We then present and discuss the measuring results of the performance costs generated by ads. We illustrate the quantified concern levels from users for the performance costs of ads for the 20 subject apps. Finally, we exhibit the correlation analysis results of the two factors.

Dataset of Subject Apps for Performance Cost Analysis

In our study, we select 20 popular apps from Google Play as the subjects based on the following four criteria: (1) they are selected from different categories - to ensure the generalization of our results; (2) they are apps containing ads; (3) they have a large number of reviews - indicating that user feedback can be sufficiently reflected in the reviews; and (4) they can be convertible to no-ads versions - for measuring the costs caused by ads.

To collect apps that satisfy the first criterion, we randomly search the top 20 apps in each of the categories (except games and family apps) on Google Play. Since Google Play provides the number of reviews and declaration about ads, we extract apps with more than 10,000 reviews and with ads contained. To satisfy the last criterion, we convert these apps to no-ads versions based on

`Xposed` [194] in a random order and then inspect whether the ads had been successfully removed. Finally, we choose 20 subjects for our experiment analysis. Their details are illustrated in Table 7.5.

In Table 7.5, we list the category, app name, package name, version, number of reviews and overall rating for each subject app. The subject apps belong to four different categories. We crawl total 34,455 reviews published from December, 2016 to April, 2017 for the 20 apps. The reviews are large enough for review analysis [52], which can effectively capture the user experience. Moreover, since the *word2vec* model used in Section 7.2.3 usually requires large datasets to achieve better performance [121], we also collect 4,007,628 reviews from other apps to enrich the training data of *word2vec*. The trained *word2vec* model can be found in our public repository [178].

Usage Trace

We introduce the selection of volunteer users and collected usage traces in this subsection.

Volunteer Users: For rendering the viewing traces of ads various, 17 users are selected from different genders (six females and 11 males), and distributed in different age groups (six of them are aged at 18-25, ten at 25-30, and one at 30-35). All the participants selected satisfy the following criteria: 1) they interact with apps for more than 30 minutes daily - indicating that the users are familiar with using mobile apps; 2) they have experience using apps of different categories - considering the multi-categories of the subject apps; and 3) they are willing to spend time on our experiments - implying that they will take patience to execute the apps ***according to their usual habits***. We invite them to exercise the functionalities of the 20 subject apps according to their own usage habits. As it is difficult for volunteers to complete interacting with all the apps at once, collecting the usage traces of one volunteer cost $1 \sim 2$ days.

Collected Interaction Duration: The average interaction time

Table 7.5: Variables and Definitions

Category	ID	App Name	Package Name	Version	# Reviews	Overall Rating
Weather	A1	RadarNow!	com.usnaviguide.radar_now	6.3	2,346	4.4
	A2	Transparent clock & weather	com.droid27.transparentclockweather	0.99.02.02	918	4.3
	A3	Weather Underground: Forecasts	com.wunderground.android.weather	5.6	4,584	4.5
	A4	AccuWeather	com.accuweather.android	4.6.0	8,691	4.3
Productivity	A5	QR & Barcode Scanner	com.gamma.scan	1.373	297	4.3
	A6	Advanced Task Killer	com.rechild.advancedtaskkiller	2.2.1B216	358	4.4
	A7	Super-Bright LED Flashlight	com.surpax.ledflashlight.panel	1.1.4	1,661	4.6
	A8	iTranslate - Free Translator	at.nk.tools.iTranslate	3.5.8	242	4.4
	A9	AVG Cleaner for Android phones	com.avg.cleaner	3.7.0.1	494	4.3
Health & Fitness	A10	Pedometer	com.tayu.tau.pedometer	5.19	2,024	4.4
	A11	Pedometer & Weight Loss Coach	cc.pacer.androidapp	2.17.0	1,576	4.5
	A12	Period Tracker	com.period.tracker.lite	2.4.4	1,332	4.5
	A13	Alarm Clock Plus*	com.vp.alarmClockPlusDock	5.2	577	4.4
	A14	Daily Ab Workout FREE	com.tinymission.dailyabworkoutfree1	5.01	25	4.4
	A15	Map My Ride GPS Cycling Riding	com.mapmyride.android2	17.2.1	408	4.4
	A16	Calorie Counter - MyFitnessPal	com.myfitnesspal.android	6.5.6	2,267	4.6
News & Magazines	A17	BBC News	bbc.mobile.news.ww	4.0.0.80	9,693	4.3
	A18	Fox News	com.foxnews.android	2.5.0	4,163	4.5
	A19	NYTimes - Latest News	com.nytimes.android	6.09.1	71	3.8
	A20	Dailyhunt (Newshunt) News	com.eterno	8.3.17	1,452	4.3

for the apps ranges from 14 seconds to 2.48 minutes. Short interaction spans may be attributed to the simple functionality provided by some apps. For example, the app "com.rechild.advancedtas- kkiller" mainly supports service killing by clicking one button on the home page, which costs about 23 seconds on average according to our records. At least 70% apps are executed for more than one minutes on average, and only one app ("com.gamma.scan") is executed with less than 20 seconds.

Reproduction of Usage Scenarios: For each app, we measure 102 times[4] by repeating both the with-ads version and no-ads version three times. The average values are calculated to alleviate noise. Note that even though running tools, such as `top` and `Xposed`, can affect mobile performance, the effects could be consistent on both versions (with-ads and no-ads) [74] and can thus be ignored in our cost measurement. Overall, we measure the 20 subject apps 2,040 times[5] in total. The whole measurement process lasted for more than one month.

Result of Cost Measurement

For each subject app, we measure the four types of performance costs (*i.e.*, memory, GPU, network and battery consumption) for both with-ads and no-ads versions. Figure 7.4 depicts the costs of the 20 apps, with blue bars indicating the memory costs of the no-ads versions and orange bars representing the ad costs. According to the figure, all the with-ads versions consume more performance cost than their no-ads versions. The memory increase ranges from 5.9% (A16) to 46.4% (A6), with an average of 25.2%. For CPU cost, ads in the subject apps consume 1.0% to 12.0% with respect to occupation rate, with median cost at 7.4%. This indicates that mobile ads indeed influence the device storage, which is consistent

[4] $102 = 17 \times 6$, where 17 is the number of volunteer users and six denotes the total measuring times for both the with-ads and no-ads versions of an app.

[5] $2040 = 102 \times 20$, where 20 denotes the number of subject apps.

with the results by Gui *et al.* [74].

Table 7.6 shows the statistics of all measured performance costs for the 20 subjects, with the average increase rate and corresponding deviation (which represents the cost increase variations among the subject apps). Network usage has the most remarkable increase (113.9%) on average. The distinct cost increase (s.d. at 108.9%) of network usage may be attributed to the ads-oriented design of some apps. CPU costs experience a modest increase (6.9% on average). Moreover, the growth in battery drainage is also noteworthy, with the average increase at 17.7% and deviation at 11.9%. Heavy performance costs may ruin user experience and drive users to uninstall the apps, which is the reason why developers and researchers pay attention to the performance costs of ads.

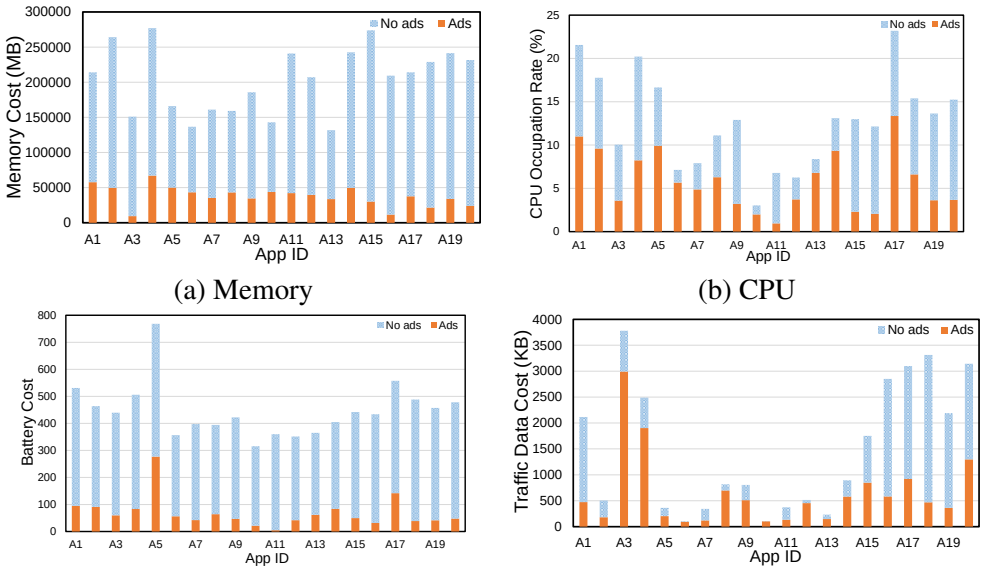


Figure 7.4: RQ3: Performance consumption of with-ads (in orange) and no-ads versions (in blue).

We further observe whether statistically significant differences exist between performance costs of with-ads versions and those of no-ads versions. We first check the distributions of each type of measured performance costs by the Shapiro-Wilk test [165].

Table 7.6: Average and standard deviation of the increase rate of performance cost when comparing with-ads version with the no-ads version.

Cost Type	Memory	CPU	Network	Battery
Average	25.2%	6.9%	**113.9%**	17.7%
Standard Deviation	12.5%	3.7%	**108.9%**	11.9%

The Shapiro-Wilk test is a typical test of normality of which the null hypothesis is that the input samples come from a normally distributed population. If the p-value computed by the Shapiro-Wilk test is smaller than 0.05, we achieve that the input distribution is significantly different from the normal distribution. Table 7.7 lists the p-value results of Shapiro-Wilk test for different performance cost types. We can discover that except for the traffic cost of with-ads versions, all the other measured costs render normal distributions. Therefore, for memory, CPU, and Battery costs, we use the *paired t-test* [87] for comparing the distributions between with-ads and no-ads versions, and use the Wilcoxon signed-rank test for analyzing the traffic costs. The paired t-test is a statistical test to determine whether the mean difference between paired observations is zero, with the p-value less than 0.05 indicating the difference between the two paired inputs is significant. We use paired t-test for costs of memory, CPU, and Battery, because the subject apps may have different cost values for with-ads and no-ads versions, and the differences between pairs are normally distributed. The Wilcoxon signed-rank test is a paired version of the Wilcoxon rank-sum test we used in Section 7.3.1.

Finding: **Performance costs of with-ads versions are significantly larger than those of no-ads versions.** Figure 7.5 illustrates the comparison on the performance costs of with-ads and no-ads versions. The p-values in paired t-test and Wilcoxon signed-rank tests show that the two input distributions are significantly different. The effect sizes measured by Vargha and Delaney's A_{12} are all negligible. The results indicate that versions with ads expend more performance costs, consistent with the studies in [74] and [159].

Table 7.7: Normality test of differences between measured performance costs of with-ads versions and no-ads versions. The p-value$<$0.05 means the differences are not normally distributed.

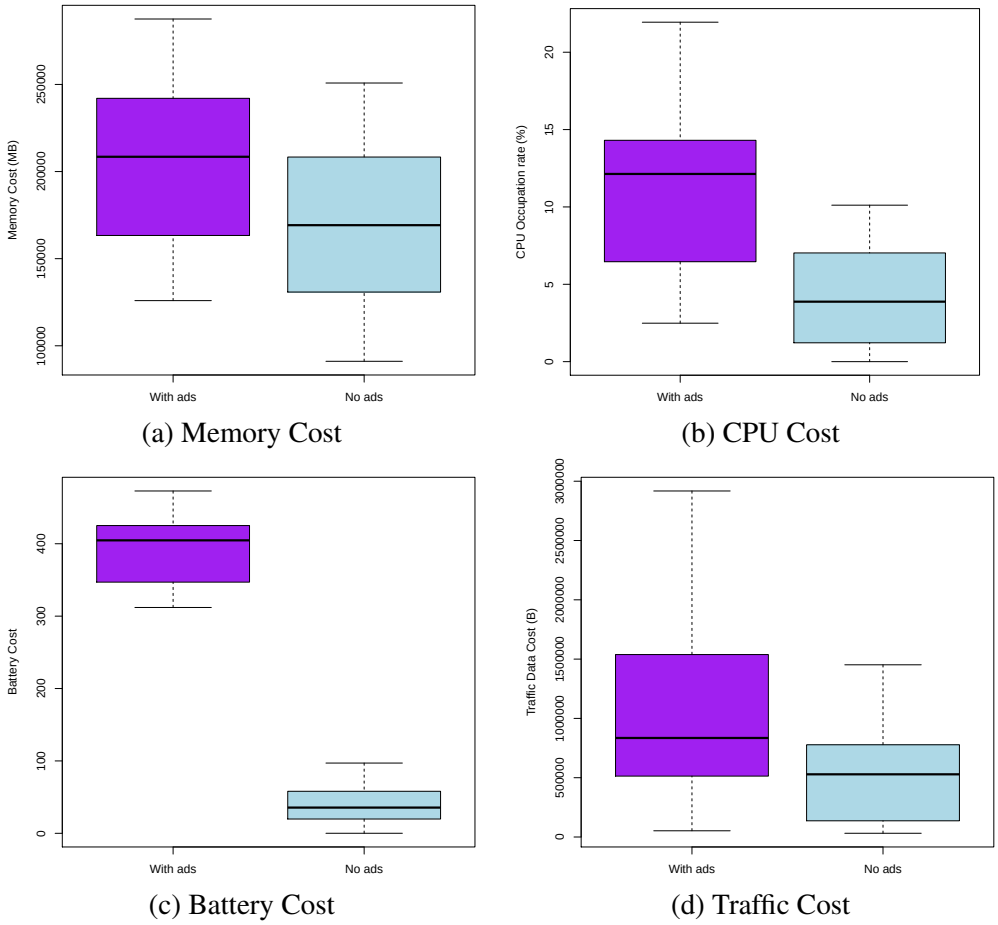
Cost Type	Memory	CPU	Battery	Traffic
p-value	0.666	0.116	0.429	**0.001**

Result of Review Analysis

Based on the methods proposed in Section 7.2.1, we extract 3,130 2-gram terms (PMI=5.0) and 5,134 3-gram words (PMI=3.0). We train the *word2vec* model on 4,042,083 reviews which include the reviews of the 20 subject apps and 4,007,628 reviews from other apps. The dimensions of the output term vectors are defined as 200, with other parameters specified by Mikolov *et al.* [121]. For each type of performance costs, we identify cost-related terms based on their cosine similarities to the target words, such as "memory", "cpu", "network", and "battery". During the experiments, we first extract the 50 most similar terms for each cost type, and then build the performance-related dictionaries after manually filtering out noise terms (illustrated in Table 7.8). To ensure that user reviews are specific to subject app versions, we select the reviews posted by users within two months[6] after the corresponding version release. Following the methods in Section 7.2.3, we calculate users' concerns over the performance costs of ads. In the following, we present the user concern analysis on the 20 subject apps.

We illustrate the results of users' concerns about the performance costs in Figure 7.6, with the blue bars and orange bars denoting the measured values for no-ads and with-ads versions respectively. For the 20 subjects, users express different levels of concerns about the memory overhead of the in-app ads. For example, for the memory cost, A2 receives the most complaints about ads among all the subject apps, with an obvious increase of 35.9% compared with the no-ads version. By inspecting A2, we discover that in-app ads can

[6]The period is defined following the previous work [56].



(a) Memory Cost (b) CPU Cost

(c) Battery Cost (d) Traffic Cost

Figure 7.5: Performance cost distributions for with-ads (in purple) and no-ads versions (in light blue).

occupy almost the whole screen space, especially with one banner in the top and one rectangle ad appearing in the middle when sliding downward. Interestingly, we find that 15 (75%) apps receive zero negative feedback about the memory costs of ads, such as A1. This implies that in most cases, users tend to be insensitive to the memory costs caused by in-app ads.

By observing the increase rate of quantified user concerns about all performance costs (shown in Table 7.9), we identify that memory costs have the largest rate of growth in user concerns (6.3% on average) and the most obvious deviation (17.0%) among the 20 apps. However, users express the least concerns about network costs,

Table 7.8: Performance-related dictionaries.

Cost Type	#Terms	Related Term
Memory	19	ram memory, storage, storage space, memory space, space, internal memory, ram, internal storage, internal space, disk space, gb, battery power, extra space, ram memory, unnecessary space, capacity, mb, valuable space, precious space
CPU	17	cpu, processor, gpu, cpu usage, laggy, slowly, too slow, incredibly slow, extremely slow, sluggish, painfully slow, terribly slow, take age, slower, slower than before, lag, fast
Network	12	network connection, data connection, wifi connection, network signal, wifi, wifi network, wifi signal, internet connectivity, wireless connection, 4g connection, internet connection, wireless network
Battery	14	battery life, battery power, batery, batt, battery drain, battery usage, battery rapidly, battery dry, battery overnight, battery juice, batterie, battery excessively, battery life, drain battery

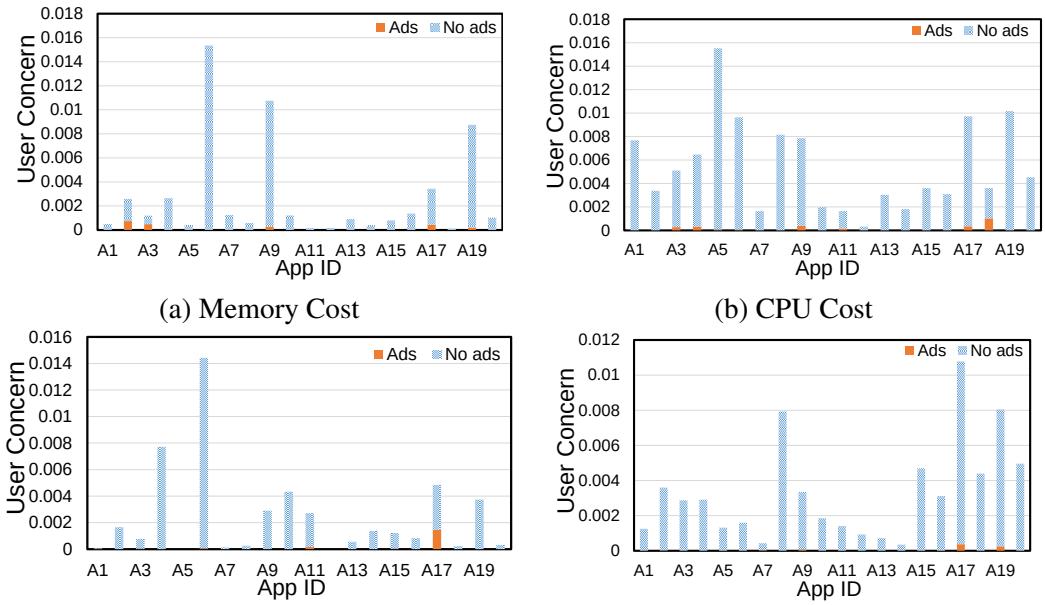
with the increase rate averaging at 0.9% and a deviation of 1.9%. Such an observation is different from what we have discovered in Table 7.6, where network costs exhibit the highest increase among all the performance costs. We find that 15/20, 12/20, 15/20, and 15/20 of the subject apps do not receive any complaints from users regarding the cost of memory, CPU, battery, and traffic, respectively. We guess that users may perceive different types of performance costs differently. We then conduct correlation analysis to explore there are strong correlations between user concerns and performance costs of ads.

Table 7.9: Increase rate of quantified user concerns about performance costs.

Cost Type	Memory	CPU	Network	Battery
Average	**6.3%**	3.5%	0.9%	2.7%
Standard Deviation	**17.0%**	8.9%	1.9%	9.6%

Correlation between Ad Costs and User Concerns

The correlations between performance costs and corresponding user concerns are illustrated in Table 7.10. Almost all the PCC results indicate that their linear correlations are weak, especially for memory



(a) Memory Cost

(b) CPU Cost

(c) Battery Cost

(d) Traffic Cost

Figure 7.6: Quantified user concerns about different performance cost types of the 20 subject apps.

usage which represents nearly no correlation with the quantified user concerns (with PCC score $r_p = -0.132$). The only one performance type that presents strong correlation with the quantified user concern is battery cost, with $r_p = 0.534$ and $p = 0.015 < 0.05$.

The results of PCC are consistent with those of SRC, where user concern shows a strongly increasing trend with more battery consumed ($p = 0.0009 \ll 0.05$). This allows us to achieve that users care most about the battery cost among all the performance cost types. We attribute this to that the consumption of battery is more sensible than other costs to users, and therefore more battery costs tend to cause more unfavorable reviews.

We also observe the negative correlation between network cost and the corresponding user concern with respect to both PCC and SRC analysis. This means that more network costs could possibly bring better user experience. This might be against our common sense. We attribute this to the ubiquity of WiFi leading to fewer

concerns about traffic consumed. According to [180], over 90% of users choose WiFi connections when using smartphones. We therefore achieve that the network consumption of ads may not be concerned to users.

For CPU costs, the PCC ($r_p = 0.166$) and SRC ($r_s = 0.213$) scores display weak correlations with user concerns. The result is predictable, as users may not perceive the CPU cost on their mobile phones, and would generally think the crash or laggy performance is caused by mobile systems or app-specific functionalities. We conclude that the effect of CPU consumption on users may be weak. Note that since our data are not time-series, causal impact analysis [116, 51] is not applicable in our situation. Moreover, our correlation analysis is applicable and convincing to determine the correlations between the two factors.

Table 7.10: Correlation test result between performance costs of ads and user concerns.

Cost Type	Memory		CPU		Network		Battery	
	r	p[3]	r	p	r	p	r	p
PCC[1]	0.132	0.578	0.166	0.482	-0.281	0.229	0.534	**0.015**
SRC[2]	0.372	0.105	0.213	0.366	-0.127	0.591	0.679	**0.0009**

[1,2] The absolute values of the PCC/SRC scores r represent very weak correlations if $|r| < 0.2$, weak correlation if $0.2 \leq |r| < 0.4$, moderate correlations if $0.4 \leq |r| < 0.6$, strong correlations if $0.6 \leq |r| < 0.8$, and very strong correlation if $|r| \geq 0.8$ [59].

[3] $p < 0.05$ indicates that the correlation is statistically significant.

Insight 5: *Users are most concerned about the battery cost of ads among all the four cost types. They tend to pay little attention to the memory and CPU cost of ads. Network cost of ads is least cared by users.*

7.4 Case Study

To validate whether our insights are endorsed by developers, we conduct a survey of engineers from Microsoft. We collect 87 survey feedback in total. We focus on analyzing the answers of

those participants who answered that they have app development experience (34/87). The participants include seven staff developers, 24 interns, and three researchers. 15 (44.1%) of them have designed more than one apps (for Android, iOS, or Windows Phone Systems) and 19 (55.9%) of them have experience in designing one app. The user study is conducted through an online questionnaire, which consists of four questions: two questions on the participant's background, one question as a transitional inquiry and about in-app advertising, and one question (including five multiple-choice questions) for assessing of our advertising insights 1∼5.

The insights are assessed via three metrics: rationality, usefulness, and novelty, where novelty indicates whether the suggestions are rarely noticed before but instructive. Insights 1∼5 correspond to the five small questions, with results in Figure 7.7. As the results indicate, more than half respondents of agree with the rationality of the insights, with Insight 2 (*i.e.*, avoiding pop-up ads) having the highest endorsement (85.3%). Also, these insights are deemed useful by 26.5%, 23.5%, 32.4%, 38.2%, and 32.4% of them, respectively. Interestingly, the novelty of Insight 5 (*i.e.*, users tend to pay little attention to the memory/CPU/netowrk cost of ads) is recognized by 20.6% of participants. In addition, 91.2% (31/34) of participants give positive feedback and consider these insights to be either rational, useful, or novel. Overall, our insights cannot only inspire developers, but also provide guidance on in-app advertising. These results strengthen the validation of our suggestions.

7.5 Summary

In the chapter, we explore the effects of in-app ads on user experience by answering three research questions, *i.e.*, can in-app ads adversely impact user behaviors towards apps, what users are actually concerned about in-app ads, and how the performance costs of ads affect user opinions. Our findings are not only complementary

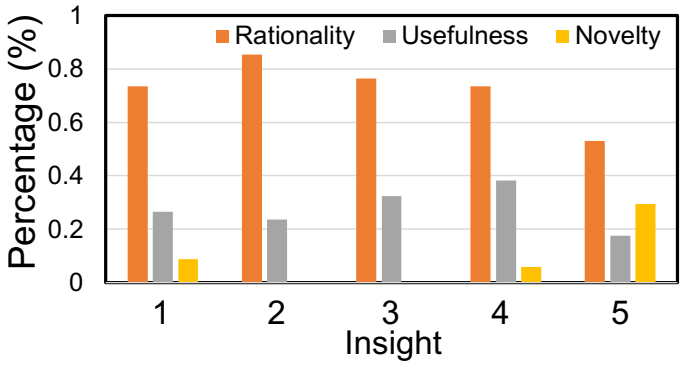


Figure 7.7: Insight validation. Insight 1 suggests that developers should choose ad SDKs with good performance in recommending relevant ads to users. Insight 2 is about avoiding pop-up ads. Insight 3 is related to avoiding full-screen ads and providing obvious closing symbols in ads. Insight 4 suggests shortening the compulsory video ads. Insight 5 summarizes that users are most concerned about the battery cost of ads, and tend to pay little attention to ads' memory, CPU, and network cost.

to previous work in this area, but also provide developers with actionable and valuable suggestions on in-app advertising. In future work, we will extend our experiments by involving much more real apps, and design an automated system for app developers to evaluate the feasibility and friendliness of in-app ads.

□ **End of chapter.**

Chapter 8

Conclusion and Future Work

In this chapter, we summarize the main contributions of this thesis and provide several interesting future directions.

8.1 Conclusion

User review mining is critical for understanding the user experience with apps and facilitating app development. However, in many cases, traditional methods involve much manual labor for filtering non-informative reviews out and labeling user intention of each review, so the application scenario of these methods is limited practically. In this thesis, we have developed novel methods to analyze online user reviews automatically for assisting release planning. We further analyze the characteristics of reviews from different platforms and user concerns about in-app advertising.

In particular, in Chapter 3, we propose an automated tool, namely AR-Tracker, for ranking informative reviews. The review-ranking method is calculated based on ranked topics, so the non-informative reviews can be avoided in the top list. Experimental evaluation on four popular apps indicates the effectiveness of our tool in review ranking. We also demonstrate a case study that shows tracking app aspects reflected in top-ranked reviews can help developers identify important app issues.

In Chapter 4, we propose an issue-prioritizing framework, called PAID, to rank phrase-level app issues automatically and accurately. PAID traces the quantitative changes of app issues over consecutive release versions. Specifically, we design a rule-based method to capture meaningful phrases, and two-layer filtering method to remove non-informative reviews. We label each topic with the most semantically relevant phrases, and track the changes in number along with app versions. We adopt official changelogs as ground truth, and the evaluation results indicate the effectiveness of PAID in prioritizing important app issues.

In Chapter 5, we design an emerging issue detection framework, namely IDEA, for identifying newly-appeared and suddenly-increasing app issues. We propose an adaptively online topic modeling method, named AOLDA, to capture topic distributions in review streams. Then an automatic topic interpretation method labels each topic with most relevant and poorly-rated phrases. Experimental evaluation on six popular apps, distributed on two app platforms, shows the effectiveness of IDEA. IDEA can be easily applied to text-based online detection tasks and report emerging issues timely. Industrial practice also validates the efficiency of IDEA in release planning.

In Chapter 6, we conduct an empirical study to analyze the issue distributions on different app platforms for the same app. We propose a framework, namely CrossMiner, to automatically retrieve relevant keywords given a word-level or phrase-level app issue. We aim at discovering the differences and similarities of app issues on three popular app platforms, *i.e.*, Google Play, App Store, and Windows Store. Based on the identified issue distributions, app developers can design and arrange the testing cases more efficiently for different platforms. To our best knowledge, CrossMiner is the first framework proposed to explore app issues on different platforms from users' perspective. The experimental study also verifies that our framework can reflect the user concerns accurately.

In Chapter 7, we aim at exploring the impact of in-app ads on user experience. We study whether in-app ads adversely impact user behaviors towards apps, analyze major user concerns delivered by user reviews, and observe whether more performance costs can cause more user concerns. Our findings are not only complementary to previous work, but also provide developers with actionable suggestions on designing in-app ads.

In summary, we design novel methods for analyzing user reviews with less manual labor and mining useful information to assist mobile app development (*e.g.*, design, testing, and updating). Specifically, our proposed methods can effectively rank informative reviews, extract important phrase-level app issues, identify abnormal app issues, discover issue distributions cross platforms, and capture user-concerned issues about in-app ads. We evaluate our research results based on large experimental datasets, with partial verification from industrial practice or user survey.

8.2 Future Work

Automatic user review analysis has been widely studied in recent years, and it is a promising research topic. Although we have proposed a number of novel frameworks that advance the state-of-the-art solutions or assist mobile app development from a different perspective, there are still many interesting research directions which are considered as future work.

Understanding User Reviews with Knowledge from User Forum

Mobile app developers are promoted to ensure user experience steadily due to furious app competition and constantly-updated user requirements. Since app reviews (*i.e.*, user feedback) can reflect current app bugs and requested features from users, exploring such source for facilitating development has been gaining much attention

from both academic and industrial communities. However, user forums, which are online peer-to-peer support and generally contain app issues provided by users in more detail, have never been studied.

Utilizing information from user forums is challenging. On one hand, the number of questions on user forums is limited and imbalanced. On the other hand, the quantity of apps with public user forums provided is not large enough, so the application scenarios of this task could be restricted.

To fill this gap, we plan to conduct comprehensive exploration on user forums, and study how to exploit the information from user forms to comprehend app reviews. Specifically, we first analyze whether we can classify user reviews according to questions and question tags on user forums, as the question tags are usually more app-specific and function-based. Then we explore whether we can transfer knowledge from user forum of one app to understand user reviews of other apps with similar functionalities. Finally, we will exhibit several possible application scenarios of our study, such as automatic user forum generation.

Automatic User Review Reply

This task is based on accurate user intention mining. Positive ratings and reviews can encourage customers to download or purchase your app. In practice, developers can establish a better user experience by delivering great responses to user reviews and helping those who have requests or problems in the apps. Although industry usually adopts rule-based methods for automatic reply, there still exists some difficulties in timely and accurate reply. For example, the predefined rules may not cover the newly- presented issues immediately, and sometimes users have to switch to human customer service.

Automatic review reply is quite challenging. First, we should recognize whether a review indeed needs developers' reply. Second, reviews with limited contexts are difficult to accurately infer the user

intention without enough ground truth. Besides, replying reviews requires sufficient background knowledge about the app versions, and also considers users' acceptance.

In the future, we plan to implement this task based on the question and answering (QA) technique in the Natural Language Processing (NLP) field. We first collect enough datasets containing user reviews, corresponding developers' replies, and app changelogs. Then we learn the rules in the collected review-reply pairs in a supervised manner. Finally, the learned rules can be utilized to automatically generate replies for specific reviews.

User Feedback based Log Prioritization for Efficient Code Localization

Appropriate logging statements and effective log parsing can assist developers pinpointing issue-related code. To capture urgent software issues, log-based problem identification methods such as KPI correlation can be efficient. However, the quantity of logs generated by popular software systems every day is tremendous, which is a challenge for effective and efficient log prioritization even with these methods. For example, the WeChat software of Tencent involves thousands of modules running on tens of thousands of servers, and can produce terabyte-level logs.

Since user feedback delivers users recent experience with software systems, it is real-time resource for developers. User feedback generally reflects existing software bugs and features to improve, and its volume is much smaller than that of logs. Therefore, mining user feedback is helpful for prioritizing logs that associate with software issues.

To this end, we plan to design a code localization framework based on user feedback and generated logs. Specifically, we will use deep learning based methods (e.g., convolutional neural networks or recurrent neural networks) to convert log sequences to embeddings. The log sequences are extracted based on effective log parsing, and can be regarded as sentences or documents. User feedback

will be online processed, during which important user-concerned issues are detected. The captured software issues will also be converted to embeddings by the typical word2vec model or other language models. Based on the issue embeddings and log sequence embeddings, we identify important logs. These logs will be utilized for code localization.

□ **End of chapter.**

Appendix A

List of Publications

1. Jichuan Zeng, Jing Li, Yan Song, **Cuiyun Gao**, Michael R. Lyu, and Irwin King. *Topic memory networks for short text classification*. The 28th International Conference on Empirical Methods in Natural Language Processing (EMNLP), 2018.

2. **Cuiyun Gao**, Jichuan Zeng, Federica Sarro, Michael R. Lyu, Irwin King. *Exploring the Effects of Ad Schemes on the Performance Cost of Mobile Phones*. The 1st International Workshop on Advances in Mobile App Analysis (A-Mobile), 2018.

3. **Cuiyun Gao**, Jichuan Zeng, David Lo, Chin-Yew Lin, Michael R. Lyu, Irwin King. *INFAR: Insight extraction from app reviews*. The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Demo track, 2018.

4. **Cuiyun Gao**, Jichuan Zeng, Michael R. Lyu, Irwin King. *Online app review analysis for identifying emerging issues*. The 40th International Conference on Software Engineering (ICSE), 2018.

5. **Cuiyun Gao**, Yichuan Man, Hui Xu, Jieming Zhu, Yangfan Zhou, Michael R. Lyu. *IntelliAd: Assisting mobile app developers in measuring ad costs automatically*. The 39th

International Conference on Software Engineering Companion (ICSE-C), 2017.

6. **Cuiyun Gao**, Hui Xu, Yichuan Man, Yangfan Zhou, Michael R. Lyu. *IntelliAd: Understanding in-app ad costs from users perspective*. arXiv:1607.03575, 2016.

7. Yichuan Man, **Cuiyun Gao**, Michael R. Lyu, Jiuchun Jiang. *Experience report: Understanding cross-platform app issues from user reviews*. The 27th IEEE International Symposium on Software Reliability Engineering (ISSRE), 2016.

8. **Cuiyun Gao**, Baoxiang Wang, Pinjia He, Jieming Zhu, Yangfan Zhou, Michael R. Lyu. *PAID: Prioritizing App Issues for Developers by Tracking User Reviews Over Versions*. The 26th International Symposium on Software Reliability Engineering (ISSRE), 2015.

9. **Cuiyun Gao**, Hui Xu, Junjie Hu, Michael R. Lyu. *ARTRacker: Track the dynamics of mobile apps via user review mining*. The IEEE Symposium on Service-Oriented System Engineering (SOSE), 2015.

10. Hui Xu, Yangfan Zhou, **Cuiyun Gao**, Yu Kang, Michael R. Lyu. *SpyAware: Investigating the privacy leakage signatures in app execution traces*. The 26th International Symposium on Software Reliability Engineering (ISSRE), 2015.

□ **End of chapter.**

Bibliography

[1] APK4Fun. http://www.apk4fun.com/.

[2] App Annie. https://www.appannie.com/en/.

[3] AppBrain. http://www.appbrain.com/.

[4] AppFigures. https://appfigures.com/getstarted.

[5] Bag of Words Meets Bags of Popcorn. https://www.kaggle.com/c/word2vec-nlp-tutorial.

[6] Changelog on Wikipedia. http://en.wikipedia.org/wiki/Changelog.

[7] Consecutive Letters. http://www.fun-with-words.com/word_consecutive_letters.html.

[8] Enchant. http://www.abisource.com/projects/enchant/.

[9] English Letter Frequency Counts. http://norvig.com/mayzner.html.

[10] Facebook Messenger users gripe and grumble in online reviews. http://www.cnet.com/news/facebook-users-share-messenger-displeasure-in-online-pool/.

[11] Introduction to Android. http://developer.android.com/guide/index.html.

[12] iOS vs. Android: Your Best Arguments. http://lifehacker.com/ios-vs-android-your-best-arguments-1334921103.

[13] Jensen Shannon divergence. https://en.wikipedia.org/wiki/Jensen-Shannon_divergence.

[14] LDA on small datasets. https://stats.stackexchange.com/questions/78926/at-what-point-does-lda-latent-dirichlet-allocation-not-make-sense-to-use.

[15] Lemmatization. http://en.wikipedia.org/wiki/Lemmatisation.

[16] Multi-tasking in iOS. https://developer.apple.com/ios/human-interface-guidelines/features/multitasking/.

[17] NLTK. http://www.nltk.org.

[18] NLTK Wordnet Synsets. http://www.nltk.org/howto/wordnet.html.

[19] Number of apps available in leading app stores as of July 2015. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[20] Pfeiffer Report 2013. http://www.diffen.com/difference/Android_vs_iOS.

[21] PMI. https://en.wikipedia.org/wiki/Pointwise_mutual_information.

[22] Punkt tokenizer. http://www.nltk.org/modules/nltk/tokenize/punkt.html.

[23] Smartphone OS Market Share. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[24] Softmax function. https://en.wikipedia.org/wiki/Softmax_function.

[25] Spotify. https://community.spotify.com/.

[26] Stemming. http://en.wikipedia.org/wiki/Stemming.

[27] Ten of the Best Cross-Platform Mobile Development Tools. http://appindex.com/blog/ten-best-cross-platform-development-mobile-enterprises/.

[28] User forum of Youtube iOS. https://productforums.google.com/forum/#!forum/youtube.

[29] What Is Mobile Testing. https://smartbear.com/learn/software-testing/what-is-mobile-testing/.

[30] What Kind Of Person Prefers An iPhone? http://www.forbes.com/sites/toddhixon/2014/04/10/what-kind-of-person-prefers-an-iphone/#2dc096983e5a.

[31] WizNote. https://www.wiz.cn/.

[32] A hand-held world: the future of mobile advertising. http://www.business.com/mobile-marketing/the-future-of-mobile-advertising/, 2017.

[33] Top 7 reasons why people uninstall mobile apps. http://cdn2.hubspot.net/hubfs/355159/10_Reasons_Why_Users_Uninstall_Your_Mobile_App.pdf, 2016.

[34] AdBlocker Reborn. `http://repo.xposed.info/module/com.aviraxp.adblocker.continued`, 2018.

[35] A. Agrawal, W. Fu, and T. Menzies. What is wrong with topic modeling? and how to fix it using search-based software engineering. *Information & Software Technology*, 98:74–88, 2018.

[36] M. S. Ahmad, N. E. Musa, R. Nadarajah, R. Hassan, and N. E. Othman. Comparison between android and ios operating system in terms of security. In *Proceedings of the 8th International Conference on Information Technology in Asia (CITA)*, pages 1–4. IEEE, 2013.

[37] L. AlSumait, D. Barbará, and C. Domeniconi. On-line LDA: adaptive topic models for mining text streams with applications to topic detection and tracking. In *Proceedings of the 8th IEEE International Conference on Data Mining, ICDM 2008, December 15-19, 2008, Pisa, Italy*, pages 3–12, 2008.

[38] Android aapt. `http://stackoverflow.com/questions/15134258/android-how-to-use-android-asset-packaging-tool`, 2013.

[39] Android top. `http://adbshell.com/commands/adb-shell-top`, 2018.

[40] Distribution of free and paid Android apps in the Google Play. `https://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps/`, 2018.

[41] Number of apps of 1st quarter 2018. `https://www.statista.com/statistics/276623/number-`

of-apps-available-in-leading-app-stores/, 2018.

[42] R. Arun, V. Suresh, C. E. V. Madhavan, and M. N. Murty. On finding the natural number of topics with latent dirichlet allocation: Some observations. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part I*, pages 391–402, 2010.

[43] A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 12–13, 2014.

[44] Z. Benenson, F. Gassmann, and L. Reinfelder. Android and ios users' differences concerning security and privacy. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013, Extended Abstracts*, pages 817–822, 2013.

[45] D. M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012.

[46] D. M. Blei, T. L. Griffiths, M. I. Jordan, and J. B. Tenenbaum. Hierarchical topic models and the nested chinese restaurant process. In *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, pages 17–24, 2003.

[47] D. M. Blei and J. D. Lafferty. Correlated topic models. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-*

8, 2005, Vancouver, British Columbia, Canada], pages 147–154, 2005.

[48] D. M. Blei and J. D. Lafferty. Dynamic topic models. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 113–120, 2006.

[49] D. M. Blei, A. Y. Ng, and M. I. Jordan. 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]*, pages 601–608, 2001.

[50] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[51] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, S. L. Scott, et al. Inferring causal impact using bayesian structural time-series models. *The Annals of Applied Statistics*, 9(1):247–274, 2015.

[52] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *36th International Conference on Software Engineering, ICSE 2014, Hyderabad, India - May 31 - June 07, 2014*, pages 767–778, 2014.

[53] T. Chen, I. Ullah, M. A. Kâafar, and R. Boreli. Information leakage through mobile analytics services. In *15th Workshop on Mobile Computing Systems and Applications, HotMobile '14, Santa Barbara, CA, USA, February 26-27, 2014*, pages 15:1–15:6, 2014.

[54] M. M. Chiang and B. G. Mirkin. Intelligent choice of the number of clusters in K-means clustering: An experimen-

tal study with different cluster spreads. *J. Classification*, 27(1):3–40, 2010.

[55] H. K. Chowdhury, N. Parvin, C. Weitenberner, and M. Becker. Consumer attitude toward mobile advertising in an emerging market: An empirical study. *International Journal of Mobile Marketing*, 1(2), 2006.

[56] A. Ciurumelea, A. Schaufelbuhl, S. Panichella, and H. C. Gall. Analyzing reviews and code of mobile apps for better release planning. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, pages 91–102, 2017.

[57] W. B. Croft, D. Metzler, and T. Strohman. *Search Engines - Information Retrieval in Practice*. Pearson Education, 2009.

[58] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[59] J. D. Evans. *Straightforward statistics for the behavioral sciences*. Brooks/Cole, 1996.

[60] Facebook ad revenue growth. `https://www.forbes.com/sites/greatspeculations/2017/11/02/ad-revenue-growth-continues-to-propel-facebook/#21db7e3d65ed`, 2017.

[61] Facebook reports first quarter 2016 results. `https://bit.ly/2p8EB9m`, 2016.

[62] X. Franch and G. Ruhe. Software release planning. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pages 894–895, 2016.

[63] 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 *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 1276–1284, 2013.

[64] 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 *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*, pages 35–45, 2015.

[65] 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*, pages 284–290, 2015.

[66] Getevent. `https://source.android.com/devices/input/getevent`, 2018.

[67] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. Reran: Timing-and touch-sensitive record and replay for android. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 72–81. IEEE, 2013.

[68] Google charts. `https://developers.google.com/chart/`, 2018.

[69] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth conference on Security and Privacy in Wireless and Mobile Networks (WISEC)*, pages 101–112. ACM, 2012.

[70] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[71] T. Grønli, J. Hansen, G. Ghinea, and M. Younas. Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox OS. In *28th IEEE International Conference on Advanced Information Networking and Applications, AINA 2014, Victoria, BC, Canada, May 13-16, 2014*, pages 635–641, 2014.

[72] 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*, pages 760–770, 2015.

[73] J. Gui, D. Li, M. Wan, and W. G. J. Halfond. Lightweight measurement and estimation of mobile ad energy consumption. In *Proceedings of the 5th International Workshop on Green and Sustainable Software, GREENS@ICSE 2016, Austin, Texas, USA, May 16, 2016*, pages 1–7, 2016.

[74] J. Gui, S. Mcilroy, M. Nagappan, and W. G. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, pages 100–110. IEEE, 2015.

[75] E. Guzman, M. El-Haliby, and B. Bruegge. Ensemble methods for app review classification: An approach for software evolution (N). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 771–776, 2015.

[76] 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, pages 153–162, 2014.

[77] M. Harman, Y. Jia, and Y. Zhang. 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*, pages 108–111, 2012.

[78] Y. Hashimoto and R. Matsushita. Heat map scope technique for stacked time-series data visualization. In *16th International Conference on Information Visualisation, IV 2012, Montpellier, France, July 11-13, 2012*, pages 270–273, 2012.

[79] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. Themeriver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9–20, 2002.

[80] V. J. Hellendoorn and P. T. Devanbu. Are deep neural networks the best choice for modeling source code? In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 763–773, 2017.

[81] A. Hindle. Green software engineering: The curse of methodology. In *Leaders of Tomorrow Symposium: Future of Software Engineering, FOSE@SANER 2016, Osaka, Japan, March 14, 2016*, pages 46–55, 2016.

[82] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. Greenminer: a hardware based mining software repositories software energy consumption framework. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 12–21, 2014.

[83] T. Hofmann. Probabilistic latent semantic analysis. In *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999*, pages 289–296, 1999.

[84] L. Hong and B. D. Davison. Empirical study of topic modeling in twitter. In *Proceedings of the 3rd Workshop on Social Network Mining and Analysis, SNAKDD 2009, Paris, France, June 28, 2009*, pages 80–88, 2010.

[85] L. Hoon, R. Vasa, J. Schneider, and K. Mouzakis. A preliminary analysis of vocabulary in mobile app user reviews. In *The 24th Australian Computer-Human Interaction Conference, OzCHI '12, Melbourne, VIC, Australia - November 26 - 30, 2012*, pages 245–248, 2012.

[86] L. Hoon, R. Vasa, J.-G. Schneider, J. Grundy, et al. An analysis of the mobile app review landscape: trends and implications. *Faculty of Information and Communication Technologies, Swinburne University of Technology, Tech. Rep*, 2013.

[87] H. Hsu and P. A. Lachenbruch. Paired t test. *Wiley Encyclopedia of Clinical Trials*, 2008.

[88] J. Huang, M. Peng, H. Wang, J. Cao, W. Gao, and X. Zhang. A probabilistic method for emerging topic tracking in microblog stream. *World Wide Web*, 20(2):325–350, 2017.

[89] 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*, pages 41–44, 2013.

[90] C. Iacob, V. Veerappa, and R. Harrison. What are you complaining about?: a study of online reviews of mobile ap-

plications. In *BCS-HCI '13 Proceedings of the 27th International BCS Human Computer Interaction Conference, Brunel University, London, UK, 9-13 September 2013*, page 29, 2013.

[91] D. Ip, K. L. Ka Keung, W. Cui, H. Qu, and H. Shen. A visual approach to text corpora comparison. In *Proceedings of the 1st International Workshop on Intelligent Visual Interfaces for Text Analysis*, pages 21–24. ACM, 2010.

[92] A. Islam and D. Z. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *TKDD*, 2(2):10:1–10:25, 2008.

[93] O. Jin, N. N. Liu, K. Zhao, Y. Yu, and Q. Yang. Transferring topical knowledge from auxiliary long texts for short text clustering. In *CIKM*, pages 775–784. ACM, 2011.

[94] P. Kanerva, J. Kristoferson, and A. Holst. Random indexing of text samples for latent semantic analysis. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 22, 2000.

[95] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. What do mobile app users complain about? *IEEE Software*, 32(3):70–77, 2015.

[96] J. Kim, C. Kim, Y. Park, and H. Lee. Trends and relationships of smartphone application services: Analysis of apple app store using text mining-based network analysis. In *Proceedings of the 4th ISPIM Innovation Symposium*, 2012.

[97] D. Kong, L. Cen, and H. Jin. AUTOREB: automatically understanding the review-to-behavior fidelity in android applications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 530–541, 2015.

[98] H. Lakkaraju, R. Socher, and C. Manning. Aspect specific sentiment analysis using hierarchical deep learning. In *NIPS Workshop on Deep Learning and Representation Learning*, 2014.

[99] C. Li, Y. Duan, H. Wang, Z. Zhang, A. Sun, and Z. Ma. Enhancing topic modeling for short texts with auxiliary word embeddings. *ACM Trans. Inf. Syst.*, 36(2):11:1–11:30, 2017.

[100] C. Li, H. Wang, Z. Zhang, A. Sun, and Z. Ma. Topic modeling for short texts with auxiliary word embeddings. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 165–174, 2016.

[101] Y. Li, D. McLean, Z. Bandar, J. O'Shea, and K. A. Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE Trans. Knowl. Data Eng.*, 18(8):1138–1150, 2006.

[102] C. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007.

[103] C. X. Lin, B. Zhao, Q. Mei, and J. Han. PET: a statistical model for popular events tracking in social communities. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 929–938, 2010.

[104] D. Lin and X. Wu. Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL (ACL)*, pages 1030–1038. Association for Computational Linguistics, 2009.

[105] Q. Lin, J. Lou, H. Zhang, and D. Zhang. idice: problem identification for emerging issues. In *Proceedings of the 38th*

International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016, pages 214–224, 2016.

[106] Y. Liu, F. Li, L. Guo, B. Shen, and S. Chen. A comparative study of android and ios for accessing internet streaming services. In *Passive and Active Measurement*, pages 104–114. Springer, 2013.

[107] Y. Liu, C. Xu, and S. Cheung. Characterizing and detecting performance bugs for smartphone applications. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 1013–1024, 2014.

[108] W. Luiz, F. Viegas, R. O. de Alencar, F. Mourão, T. Salles, D. Carvalho, M. A. Gonçalves, and L. C. da Rocha. A feature-oriented sentiment rating for mobile app reviews. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1909–1918, 2018.

[109] T. Luo, X. Jin, A. Ananthanarayanan, and W. Du. Touchjacking attacks on web in android, ios, and windows phone. In *Foundations and Practice of Security - 5th International Symposium, FPS 2012, Montreal, QC, Canada, October 25-26, 2012, Revised Selected Papers*, pages 227–243, 2012.

[110] Z. Ma, A. Sun, Q. Yuan, and G. Cong. Topic-driven reader comments summarization. In *CIKM*, pages 265–274. ACM, 2012.

[111] 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)*, pages 116–125. IEEE, 2015.

[112] 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*, pages 138–149, 2016.

[113] A. S. Manek, P. D. Shenoy, M. C. Mohan, and K. R. Venugopal. Aspect term extraction for sentiment analysis in large movie reviews using gini index feature selection method and SVM classifier. *World Wide Web*, 20(2):135–154, 2017.

[114] I. Manotas, C. Bird, R. Zhang, D. C. Shepherd, C. Jaspan, C. Sadowski, L. L. Pollock, and J. Clause. An empirical study of practitioners' perspectives on green software engineering. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, pages 237–248, 2016.

[115] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR'15*, pages 123–133, 2015.

[116] W. Martin, F. Sarro, and M. Harman. Causal impact analysis for app releases in google play. In *Proceedings of the 24th SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 435–446. ACM, 2016.

[117] 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*, 21(3):1346–1370, 2016.

[118] R. Mehrotra, S. Sanner, W. L. Buntine, and L. Xie. Improving LDA topic models for microblogs via tweet pooling and automatic labeling. In *SIGIR*, pages 889–892. ACM, 2013.

[119] Q. Mei, X. Shen, and C. Zhai. Automatic labeling of multinomial topic models. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 490–499, 2007.

[120] W. Meng, R. Ding, S. P. Chung, S. Han, and W. Lee. The price of free: Privacy leakage in personalized mobile in-apps ads. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[121] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[122] A. Mnih and K. Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS*, pages 2265–2273, 2013.

[123] Number of smartphone users worldwide. `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`, 2018.

[124] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *Proceedings of the 8th European Conference on Computer Systems (EuroSys)*, pages 267–280. ACM, 2013.

[125] Monsoon Power Monitor. `https://www.msoon.com/LabEquipment/PowerMonitor/`, 2018.

[126] R. C. Moore. On log-likelihood-ratios and the significance of rare events. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*, pages 333–340, 2004.

[127] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *AISTATS*. Society for Artificial Intelligence and Statistics, 2005.

[128] R. Mouawi, I. H. Elhajj, A. Chehab, and A. I. Kayssi. Comparison of in-app ads traffic in different ad networks. In *11th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2015, Abu Dhabi, United Arab Emirates, October 19-21, 2015*, pages 581–587, 2015.

[129] S. Nath. Madscope: Characterizing mobile in-app targeted ads. In *Proceedings of the 13th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 59–73. ACM, 2015.

[130] S. Nath, F. X. Lin, L. Ravindranath, and J. Padhye. Smartads: bringing contextual ads to mobile apps. In *The 11th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'13, Taipei, Taiwan, June 25-28, 2013*, pages 111–124, 2013.

[131] 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*, pages 552–562, 2016.

[132] M. Nayebi and G. Ruhe. Asymmetric release planning-compromising satisfaction against dissatisfaction. *IEEE Transactions on Software Engineering*, (1):1–1, 2018.

[133] A. T. Nguyen, M. Hilton, M. Codoban, H. A. Nguyen, L. Mast, E. Rademacher, T. N. Nguyen, and D. Dig. API code recommendation using statistical learning from fine-grained changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 511–522, 2016.

[134] T. Nguyen, H. W. Lauw, and P. Tsaparas. Review synthesis for micro-review summarization. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM 2015, Shanghai, China, February 2-6, 2015*, pages 169–178, 2015.

[135] K. Nigam, A. McCallum, S. Thrun, and T. M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[136] NLTK stopwords. `http://www.nltk.org/book/ch02.html`, 2015.

[137] D. D. Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. D. Lucia. Software-based energy profiling of android apps: Simple, efficient and reliable? In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, pages 103–114, 2017.

[138] P value. `https://explorable.com/p-value`, 2018.

[139] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *Proceedings of the 21st IEEE Interna-*

tional Conference on Requirements Engineering Conference (RE), pages 125–134. IEEE, 2013.

[140] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. C. Gall, F. Ferrucci, and A. D. Lucia. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 106–117, 2017.

[141] 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*, pages 291–300, 2015.

[142] S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. 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*, pages 281–290, 2015.

[143] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In *The 10th International Conference on Mobile Systems, Applications, and Services, MobiSys'12, Ambleside, United Kingdom - June 25 - 29, 2012*, pages 267–280, 2012.

[144] Pearson correlation. `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`, 2018.

[145] X. H. Phan, M. L. Nguyen, and S. Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *WWW*, pages 91–100. ACM, 2008.

[146] O. Phelan, K. McCarthy, and B. Smyth. Using twitter to recommend real-time topical news. In *RecSys*, pages 385–388. ACM, 2009.

[147] E. Platzer. Opportunities of automated motive-based user review analysis in the context of mobile app acceptance. In *Proceedings of the 2011 CECIIS*, 2011.

[148] E. Pujol, O. Hohlfeld, and A. Feldmann. Annoyed users: Ads and ad-block usage in the wild. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, pages 93–106, 2015.

[149] J. Qiang, P. Chen, T. Wang, and X. Wu. Topic modeling over short texts by incorporating word embeddings. In *Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part II*, pages 363–374, 2017.

[150] X. Quan, C. Kit, Y. Ge, and S. J. Pan. Short and sparse text topic modeling via self-aggregation. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2270–2276, 2015.

[151] D. Ramage, S. T. Dumais, and D. J. Liebling. Characterizing microblogs with topic models. In *ICWSM*. The AAAI Press, 2010.

[152] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. Riley. Are these ads safe: Detecting hidden attacks through the

mobile app-web interfaces. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[153] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50. ELRA, 2010.

[154] X. Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[155] M. Rosen-Zvi, C. Chemudugunta, T. L. Griffiths, P. Smyth, and M. Steyvers. Learning author-topic models from text corpora. *ACM Trans. Inf. Syst.*, 28(1):4:1–4:38, 2010.

[156] P. J. Rousseeuw and M. Hubert. Robust statistics for outlier detection. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 1(1):73–79, 2011.

[157] G. Ruhe. *Product Release Planning - Methods, Tools and Applications*. CRC Press, 2010.

[158] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan. Impact of ad libraries on ratings of android mobile apps. *IEEE Software*, 31(6):86–92, 2014.

[159] R. Saborido, F. Khomh, G. Antoniol, and Y. Guéhéneuc. Comprehension of ads-supported and paid android applications: are they different? In *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017, Buenos Aires, Argentina, May 22-23, 2017*, pages 143–153, 2017.

[160] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW*, pages 377–386. ACM, 2006.

[161] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984.

[162] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain, and die in app stores. In *23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015*, pages 76–85, 2015.

[163] F. Sarro, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Life and death in the app store: Theory and analysis of feature migration. *RN*, 14:12, 2014.

[164] A. Seneviratne, K. Thilakarathna, S. Seneviratne, M. A. Kâafar, and P. Mohapatra. Reconciling bitter rivals: Towards privacy-aware and bandwidth efficient mobile ads delivery networks. In *Fifth International Conference on Communication Systems and Networks, COMSNETS 2013, Bangalore, India, January 7-10, 2013*, pages 1–10, 2013.

[165] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[166] S. Sooel, K. Daehyeok, and S. Vitaly. What mobile ads know about mobile users. 2016.

[167] A. D. 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 ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 499–510, 2016.

[168] S. Soroa-Koury and K. C. Yang. Factors affecting consumers' responses to mobile advertising from a social norm theoretical

perspective. *Telematics and Informatics*, 27(1):103–113, 2010.

[169] Spearman rank correlation. `https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient`, 2018.

[170] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas. Short text classification in twitter to improve information filtering. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 841–842, 2010.

[171] Stanford POS tagging. `https://nlp.stanford.edu/software/tagger.shtml`, 2018.

[172] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, volume 10, 2012.

[173] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu. Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 565–576, 2009.

[174] Android tcpdump. `http://www.androidtcpdump.com/`, 2018.

[175] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment in short strength detection informal text. *JASIST*, 61(12):2544–2558, 2010.

[176] T. T. Thet, J. Na, and C. S. G. Khoo. Aspect-based sentiment analysis of movie reviews on discussion boards. *J. Information Science*, 36(6):823–848, 2010.

[177] K. W. Tracy. Mobile application development experiences on apple¿ s ios and android os. *Ieee Potentials*, 31(4):30–34, 2012.

[178] Trained word2vec on app reviews. `https://github.com/ReMine-Lab/IDEA/`, 2018.

[179] I. Ullah, R. Boreli, M. A. Kâafar, and S. S. Kanhere. Characterising user targeting for in-app mobile ads. In *2014 Proceedings IEEE INFOCOM Workshops, Toronto, ON, Canada, April 27 - May 2, 2014*, pages 547–552, 2014.

[180] Understanding today's smartphone user. `http://www.informatandm.com/wp-content/uploads/2012/02/Mobidia_final.pdf`, 2012.

[181] Why people uninstall apps. `http://cdn2.hubspot.net/hubfs/355159/10_Reasons_Why_Users_Uninstall_Your_Mobile_App.pdf`, 2016.

[182] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: characterizing mobile advertising. In *Proceedings of Conference on Internet Measurement Conference (IMC)*, pages 343–356. ACM, 2012.

[183] A. Vargha and H. D. Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[184] N. Viennot, E. Garcia, and J. Nieh. A measurement study of google play. In *ACM SIGMETRICS / International Confer-*

ence on Measurement and Modeling of Computer Systems, SIGMETRICS '14, Austin, TX, USA - June 16 - 20, 2014, pages 221–233, 2014.

[185] 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*, pages 14–24, 2016.

[186] 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 *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 749–759, 2015.

[187] 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*, pages 726–731, 2016.

[188] X. Wan and T. Wang. Automatic labeling of topic models using text summaries. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[189] X. Wang, C. Zhai, X. Hu, and R. Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *KDD*, pages 784–793. ACM, 2007.

[190] L. Wei, Lili, Y. Liu, and S.-C. Cheung. Oasis: Prioritizing static analysis warnings for android apps based on app user reviews. In *Proceedings of the 25th SIGSOFT International*

Symposium on Foundations of Software Engineering (FSE). ACM, 2017.

[191] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Profiledroid: multi-layer profiling of android applications. In *Proceedings of the 18th International Conference on Mobile Computing and Networking (MobiCom)*, pages 137–148. ACM, 2012.

[192] J. Weng, E. Lim, J. Jiang, and Q. He. Twitterrank: finding topic-sensitive influential twitterers. In *WSDM*, pages 261–270. ACM, 2010.

[193] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[194] Xposed. `http://repo.xposed.info/module/de.robv.android.xposed.installer`, 2018.

[195] X. Yan, J. Guo, Y. Lan, and X. Cheng. A biterm topic model for short texts. In *WWW*, pages 1445–1456. International World Wide Web Conferences Steering Committee / ACM, 2013.

[196] L. Yang, M. Qiu, S. Gottipati, F. Zhu, J. Jiang, H. Sun, and Z. Chen. Cqarank: jointly model topics and expertise in community question answering. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 99–108, 2013.

[197] J. Yin and J. Wang. A dirichlet multinomial mixture model-based approach for short text clustering. In *KDD*, pages 233–242. ACM, 2014.

[198] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smart-

phone using kernel activity monitoring. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC)*, pages 387–400, 2012.

[199] J. H. Yu. You've got mobile ads! yount customers' responses to mobile ads with different types of interactivity. *International Journal of Mobile Marketing*, 8(1), 2013.

[200] W. Zhao, J. J. Chen, R. Perkins, Z. Liu, W. Ge, Y. Ding, and W. Zou. A heuristic approach to determine an appropriate number of topics in topic modeling. *BMC Bioinformatics*, 16(13):S8, 2015.

[201] W. X. Zhao, J. Jiang, J. Weng, J. He, E. Lim, H. Yan, and X. Li. Comparing twitter and traditional media using topic models. In *Advances in Information Retrieval - 33rd European Conference on IR Research, ECIR 2011, Dublin, Ireland, April 18-21, 2011. Proceedings*, pages 338–349, 2011.

[202] B. Zhou, I. Neamtiu, and R. Gupta. A cross-platform analysis of bugs and bug-fixing in open source projects: desktop vs. android vs. ios. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, EASE 2015, Nanjing, China, April 27-29, 2015*, pages 7:1–7:10, 2015.