AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace

Ning Chen, Jialiu Lin†, Steven C. H. Hoi, Xiaokui Xiao, Boshen Zhang
Nanyang Technological University, Singapore, †Carnegie Mellon University, USA
{nchen1,chhoi,xkxiao,bszhang}@ntu.edu.sg, †jialiul@cs.cmu.edu

ABSTRACT

With the popularity of smartphones and mobile devices, mobile application (a.k.a. "app") markets have been growing exponentially in terms of number of users and downloads. App developers spend considerable effort on collecting and exploiting user feedback to improve user satisfaction, but suffer from the absence of effective user review analytics tools. To facilitate mobile app developers discover the most "informative" user reviews from a large and rapidly increasing pool of user reviews, we present "AR-Miner" — a novel computational framework for App Review Mining, which performs comprehensive analytics from raw user reviews by (i) first extracting informative user reviews by filtering noisy and irrelevant ones, (ii) then grouping the informative reviews automatically using topic modeling, (iii) further prioritizing the informative reviews by an effective review ranking scheme, (iv) and finally presenting the groups of most "informative" reviews via an intuitive visualization approach. We conduct extensive experiments and case studies on four popular Android apps to evaluate AR-Miner, from which the encouraging results indicate that AR-Miner is effective, efficient and promising for app developers.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithm and Experimentation

Keywords

User feedback, mobile application, user reviews, data mining

1. INTRODUCTION

The proliferation of smartphones attracts more and more software developers to devote to building mobile applications ("apps"). As the market competition is becoming more intense, in order to seize the initiative, developers tend to employ an iterative process to develop, test, and improve apps [23]. Therefore, timely and constructive feedback from users becomes extremely crucial for developers to fix bugs, implement new features, and improve user experience agilely. One key challenge to many app developers is how to obtain and digest user feedback in an effective and efficient manner, i.e., the "user feedback extraction" task. One way to extract user feedback is to adopt typical channels used in traditional software development, such as (i) bug/change repositories (e.g., Bugzilla [3]), (ii) crash reporting systems [19], (iii) online forums (e.g., SwiftKey feedback forum [6]), and (iv) emails [10].

Unlike the traditional channels, modern app marketplaces, such as Apple App Store and Google Play, offer a much easier way (i.e., the web-based market portal and the market app) for users to rate and post app reviews. These reviews present user feedback on various aspects of apps (such as functionality, quality, performance, etc), and provide app developers a new and critical channel to extract user feedback. However, comparing with traditional channels, there are two outstanding obstacles for app developers to obtain valuable information from this new channel. First of all, the proportion of "informative" user reviews is relatively low. In our study (see Section 5.1), we found that only 35.1% of app reviews contain information that can directly help developers improve their apps. Second, for some popular apps, the volume of user reviews is simply too large to do manual checking on all of them. For example, Facebook app on Google Play receives more than 2000 user reviews per day, making it extremely time consuming to do manual checking.

To our best knowledge, very few studies were focused on extracting valuable information for developers from *user reviews in app marketplace* [28, 21, 22]. This paper formally formulates this as a new research problem. Specifically, to address this challenging problem and tackle the aforementioned two obstacles, we propose a novel computational framework, named "AR-Miner" (App Review Miner), for extracting valuable information from raw user review data with minimal human efforts by exploring effective data mining and ranking techniques. Generally speaking, given a bunch of user reviews of an app collected during a certain time interval, AR-Miner first filters out those "non-informative" ones by applying a pre-trained classifier. The remaining "informative" reviews are then put into several groups, and prioritized by our proposed novel ranking model. Finally, we visualize the ranking results in a concise and

ICSE'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2756-5/14/05...$15.00
http://dx.doi.org/10.1145/2568225.2568263

intuitive way to help app developers spot the key feedback users have.

We validate the efficacy of AR-Miner by conducting an extensive set of experiments and case studies on user reviews of four Android apps released in Google Play. In particular, we compare the ranking results generated by AR-Miner against real app developers' decisions, and also analyze the advantages of AR-Miner over manual inspection and facilities used in a traditional channel (i.e., online forum). Our empirical results have validated the effectiveness and efficiency of AR-Miner in helping apps developers. Supplementary materials (including datasets, additional results, etc.) are publicly available at[1]. In short, this paper makes the following main contributions:

- We formulate a new problem that aims to discover the most "informative" information from raw user reviews in app markets for developers to improve their apps;
- We present AR-Miner as a novel analytic framework to tackle this new problem, which includes a new, flexible and effective ranking scheme to prioritize the "informative" reviews;
- We evaluate AR-Miner based on user reviews of four popular Android apps, in which empirical results show that AR-Miner is effective and promising.

The rest of the paper is organized as follows: Section 2 discusses related work; Section 3 gives the problem statement; Section 4 presents the AR-Miner framework; Section 5 gives empirical results; Section 6 discusses limitations and threats to validity; finally Section 7 concludes this work.

2. RELATED WORK

We group related work into three major categories, and survey the literature of each category in detail below.

2.1 App Marketplace Analysis

With the rocketing development of mobile applications, app marketplace has drawn much more attention among researchers within and outside the software engineering community. In [24], Harman et al. pointed out that, app marketplace is a new form of software repository and very different from traditional ones. They also analyzed the technical, customer and business aspects of some apps in BlackBerry World. In [33], Minelli et al. proposed to combine data extracted from app marketplace with source code to comprehend apps in depth. Linares-Vásquez et al. [31] empirically analyzed how the stability and fault-proneness of APIs used by some free Android apps relate to apps' lack of success. Chia et al. [14] analyzed the relationship between permissions and community ratings of some apps. Our work is different from the aforementioned studies, mainly because we explore different data in app marketplace, i.e., user reviews, and formulate a very different problem.

Thus far, there has been little work on mining user reviews in app marketplace. In [36], Pagano and Maalej conducted an exploratory study to analyze the user reviews crawled from Apple App Store. They studied (i) the *usage* and *impact* of feedback through statistical analysis, and (ii) the *content* of feedback via manual content analysis and frequent itemset mining. Chandy et al. [13] presented a latent model to detect "bogus" user reviews in Apple App Store. Iacob et al. [28] developed a prototype named *MARA* that uses a list of linguistic rules to automatically retrieve feature requests from online user reviews. Although the nature of data studied in the above three works is similar to ours, the techniques used or research goals are totally different.

To our best knowledge, there are only two previous studies that are **closely** related to our work. Galvis Carreño et al. [22] adapted the Aspect and Sentiment Unification Model (ASUM) proposed in [30] to automatically extract topics for requirements changes. General speaking, our work differs from their work in three major aspects. First, our work aims to discover not only requirement changes, but also other kinds of valuable information for developers (see Figure 1 in Section 3). Second, we focus on the ranking scheme of "informative" user reviews, which is not addressed in their work. Finally, our scheme consisting of an effective filtering step considerably outperforms the direct application of topic models in solving our problem (see more details in Section 5.3.1). Fu et al. [21] presented a system named *Wiscom* to analyze user reviews in app marketplace at three different levels. The so-called "Meso Level" of *Wiscom* is to uncover the reasons of user dissatisfaction and their evolution over time for an individual app, which is more related to AR-Miner; however, it suffers from two major limitations. First, it cannot discover app-specific topics by using Latent Dirichlet Allocation (LDA) [11], since it links all the user reviews from the same app together as a document. Second, it only considers *negative* reviews, thus missing topics with *positive* ratings. Unlike *Wiscom*, AR-Miner can address both limitations. Moreover, we propose a new ranking model to help app developers prioritize the "informative" user reviews.

2.2 Mining User Reviews on the Web

Our work is related to studies that focus on mining and analyzing user reviews in other kinds of marketplaces (e.g., movies, commodity goods, etc.) and social webs (e.g., news sites). However, their techniques cannot be directly applied to our problem in that (1) the objective of our problem is different, i.e., solving a software engineering problem in requirement/bug management; and (2) the characteristics of user reviews in apps stores are quite different, e.g., review styles [21], volumes, etc. Next, we discuss three most related classes of work and explain specific differences.

In the literature, there is a great deal of work that applies sentiment analysis to user reviews in different marketplaces [37]. In general, these studies aim to determine the semantic orientation of a given user review at the document [40], sentence [27] or feature level [20], whether the opinion expressed is positive or negative. There are several studies focusing on summarizing/visualizing user reviews via identifying product features and opinions [42, 7]. Compared with the aforementioned work, our work differs in that (i) we classify each user review into either "informative" or "non-informative" from the software engineering perspective, instead of "positive" or "negative" from the emotional perspective; (ii) the ultimate goal of our work is different, that is visualizing the ranking results of "informative" information.

Much work has focused on spam review filtering [41, 34, 29]. They aim to protect users and honest vendors via detecting and removing bogus user reviews. Our work differs from them mainly in two points. First, in our work, the definition of "non-informative" $\neq$ "spam" (see Section 3).

[1]https://sites.google.com/site/appsuserreviews/

Class	Type (Rule)	Real Example
Informative	Functional flaw that produces incorrect or unexpected result	None of the pictures will load in my news feed.
	Performance flaw that degrades the performance of Apps	It lags and doesn't respond to my touch which almost always causes me to run into stuff.
	Requests to add/modify features	Amazing app, although I wish there were more themes to choose from.
		Please make it a little easy to get bananas please and make more power ups that would be awesome.
	Requests to remove advertisements/notifications	So many ads its unplayable!
	Requests to remove permissions	This game is adding for too much unexplained permissions.
Non-informative	Pure user emotional expression	Great fun can't put it down!
		This is a crap app.
	Descriptions of (apps, features, actions, etc.)	I have changed my review from 2 star to 1 star.
	Too general/unclear expression of failures and requests	Bad game this is not working on my phone.
	Questions and inquiries	How can I get more points?

Figure 1: Different Types of Informative and Non-informative Information for App Developers

Second, although the filtering step in AR-Miner can help remove some types of spam reviews, our major objective is to rank the "informative" user reviews for app developers.

There also exist several pieces of work on ranking reviews on the social web. For example, Hsu et al. [26] applied Support Vector Regression to rank the reviews of a popular news aggregator Digg. Dalal et al. [18] explored multi-aspects ranking of reviews of news articles using Hodge decomposition. Different from both works, our work aims to rank the reviews according to their importance (not quality) to app developers (not users) from the software engineering perspective. Besides, we propose a completely different ranking model in solving our problem.

2.3 Mining Data in Traditional Channels

Our work is also related to studies that apply data mining (machine learning) techniques on data stored in traditional channels to support developers with the "user feedback extraction" task. Specifically, the first category of related work in this field is to address problems in bug repositories [38, 9, 8, 25]. For example, Sun et al. [38] proposed a discriminative approach to detect duplicate bug reports. Anvik et al. [9] compared several classification algorithms for solving the bug assignment problem. Antoniol et al. [8] developed a machine learning approach to distinguish bugs from non-bugs. In addition, another category of related work is to solve problems in other traditional channels (e.g., request repositories [16, 15], emails [10], crash reporting systems [19]). For example, Cleland-Huang et al. [15] proposed a machine learning approach to categorize product-level requirements into pre-defined regulatory codes. Dang et al. [19] developed an approach based on similarity measures to cluster crash reports. Bacchelli et al. [10] applied a Naive Bayes classifier to classify email contents at the line-level.

Compared with the previous studies in this area, our work differs in that we formulate and solve a brand new problem in a new channel with its distinct features.

3. THE PROBLEM STATEMENT

The "user feedback extraction" task is extremely important in bug/requirement engineering. In this paper, we formally formulate it as a new research problem, which aims to facilitate app developers to find the most "informative" information from large and rapidly increasing pool of raw user reviews in app marketplace.

Consider an individual *app*, in a time interval $\mathcal{T}$, it receives a list of user reviews $\mathcal{R}^*$ with an attribute set $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$, and $r_i = \{r_i.A_1, r_i.A_2, \ldots, r_i.A_k\}$ is the i-th review instance in $\mathcal{R}^*$. Without loss of generality, in this work, we choose $\mathcal{A} = \{Text, Rating, Timestamp\}$, since these are the common attributes supported in all mainstream app marketplaces. Table 1 shows an example of $\mathcal{R}^*$ with t review instances. In particular, we set the $Text$ attribute of r_i at the **sentence** level. We will explain how to achieve and why we use this finer granularity in Section 4.2.

Table 1: Example of A List of User Reviews $\mathcal{R}^*$, R = Rating, TS = Timestamp

ID	Text	R	TS
r_1	Nice application, but lacks some important features like support to move on SD card.	4	Dec 09
r_2	So, I am not giving five star rating.	4	Dec 09
r_3	Can't change cover picture.	3	Jan 18
r_4	I can't view some cover pictures even mine.	2	Jan 10
r_5	Wish it'd go on my SD card.	5	Dec 15
...	...	...	...
r_t	...	...	...

In our problem, each r_i in $\mathcal{R}^*$ is either "**informative**" or "**non-informative**". Generally, "informative" implies r_i contains information that app developers are looking to identify and is potentially useful for improving the quality or user experience of apps. We summarize different types of "informative" as well as "non-informative" information in Figure 1 (one or two examples for each type). For example, r_1, r_3, r_4 and r_5 shown in Table 1 are "informative", since they report either bugs or feature requests, while r_2 is "non-informative", as it is a description of some user action, and developers cannot get constructive information from it.

Remark. The summarization shown in Figure 1 is not absolutely correct, since the authors are not app developers. In fact, even for real app developers, no two people would have the exact same understanding of "informative". This is an internal threat of validity in our work. To alleviate this threat, we first studied some online forums (e.g., [6]) to identify what kinds of information do **real app developers** consider as constructive, and then derived the summarization shown in Figure 1 based on the findings.

Generally, given a list of user reviews $\mathcal{R}^*$ of an *app* (e.g., the one shown in Table 1), the goal of our problem is to filter out those "non-informative" reviews (e.g., r_2), then (i)

group the remaining reviews based on the topics they are talking about, e.g., $\{r_1, r_5\}$ are grouped because they both talk about feature request related to "SD card"; and (ii) identify the relative importance of different groups and reviews in the same group (e.g., the relative importance of r_1 and r_5), and finally present an intuitive visualized summarization to app developers.

4. OUR FRAMEWORK

In this section, we first give an overview of our proposed AR-Miner framework to address the problem stated in Section 3, and then present each step of our framework in detail.

4.1 Overview

Figure 2 presents an overview of AR-Miner, which consists of five major steps. The first step preprocesses the raw user review data into well-structured format to facilitate subsequent tasks (Section 4.2). The second step applies a pre-trained classifier to filter out "non-informative" reviews in $\mathcal{R}^*$ (Section 4.3). The third step groups the remaining "informative" reviews in such a way that reviews in the same group are more semantically similar (Section 4.4). The fourth step (the focus of this paper) sorts (i) groups, and (ii) reviews in each group according to their level of importance by using our novel ranking model (Section 4.5). In the last step, we visualize the ranking results and present an intuitive summary to app developers (Section 4.6).

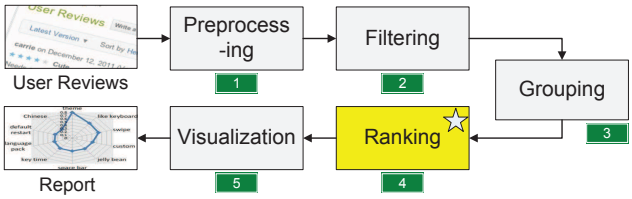


Figure 2: Overview of the AR-Miner framework. We focus on tackling the "Ranking" Step.

4.2 Preprocessing

The first step of AR-Miner preprocesses the collected raw data by (i) converting the raw user reviews into sentence-level review instances, and then (ii) preprocessing the *Text* attribute of the review instances.

The format of raw user reviews varies with different app marketplaces. As mentioned in Section 3, in this work, we choose $\mathcal{A} = \{Text, Rating, Timestamp\}$. Figure 3 shows a real example of a raw user review that contains these three attributes. The *Text* attribute of a raw user review often consists of more than one sentence. In this work, we split *Text* into several sentences via a standard sentence splitter provided by LingPipe [4]. For each sentence, we generate a review instance r_i with *Rating* and *Timestamp* equal to the values of the corresponding raw user review. For example, the raw user review shown in Figure 3 is converted into two sentence-level review instances shown in Table 1 (r_1 and r_2). We choose the **sentence-level** granularity because within a raw user review some sentences can be "informative" (e.g., sentence 1 shown in Figure 3) and some sentences are not (e.g., sentence 2). Thus, this finer granularity can help distinguish "informative" with "non-informative" information more accurately.

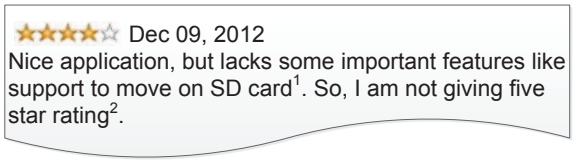


Figure 3: An Example Raw User Review

Further, we preprocess the *Text* attribute of review instances. We first tokenize the text, and then remove all non-alpha-numeric symbols, convert words to lowercase and eliminate extra whitespace along with stop words/rare words. Next, the remaining words are stemmed to their root form. Finally, we remove review instances that become empty as a result of the above processing.

4.3 Filtering

The preprocessing step generates a review database $\mathcal{R}^*$ (e.g., as shown in Table 1). In this step, our goal is to train some classifier that can automatically filter out "non-informative" reviews from $\mathcal{R}^*$.

First, we introduce the class label set used in our problem. As described in Section 3, we have two unique class labels {informative, non-informative}, where "informative" implies that the review is constructive/helpful to app developers, and "non-informative" means that the review contains no information that is useful for improving apps. We use the rules (types) summarized in Figure 1 to assign class labels to review instances. In particular, we solve some ambiguous cases. For example, we classify "too general/unclear expression of failures and requests" as "non-informative" (e.g., "It doesn't work", "It needs more update", and etc.).

To eliminate "non-informative" review instances, we need to apply a machine learning algorithm to build some classifier on the historical training data. In this work, we simply adopt a well-known and representative semi-supervised algorithm in machine learning, i.e., Expectation Maximization for Naive Bayes (EMNB) [35]. The most important reason we choose EMNB is that it suits our problem well. In our problem, we can get a mass of unlabeled data almost freely, but labeling training data is time consuming and labor intensive. Compared with supervised algorithms, EMNB can use a small amount of labeled data (thus less human effort) along with plenty of unlabeled data to train a fairly good classifier (see our comparisons in Section 5.5.1). Besides, NB often outperforms other machine learning algorithms in text classification [12] and has been widely used in other software engineering problems [10, 39]. Finally, NB provides a nice posterior probability for the predicated class, which is useful in the ranking step (See Section 4.5.3).

Once the classifier is built, it can be applied to filter future unlabeled user reviews. Table 2 shows a possible good result (denoted as $\mathcal{R}$, $n \leq t$) after filtering $\mathcal{R}^*$ shown in Table 1, where "non-informative" review instances are eliminated (r_2), and "informative" ones are preserved (r_1, r_3, r_4 and r_5). The last column "P" of Table 2 indicates the probability of the review instance belongs to the "informative" class.

4.4 Grouping

This step is to partition the remaining review instances ($\mathcal{R}$) into several groups such that the *Text* of review instances in a group is more semantically similar to each other than the *Text* of review instances in other groups.

Table 2: $\mathcal{R}$, A Possible Good Result after Filtering, R = Rating, TS = Timestamp, P = Probability

ID	Text	R	TS	P
r_1	Nice application, but lacks some important features like support to move on SD card.	4	Dec 09	0.8
r_3	Can't change cover picture.	3	Jan 18	0.9
r_4	I can't view some cover pictures even mine.	2	Jan 10	0.9
r_5	Wish it'd go on my SD card.	5	Dec 15	0.9
...	...	...	...	
r_n	...	...	...	

In general, there are two categories of unsupervised techniques that can be applied to the grouping task. The first category is *clustering* (e.g., K-means [32]), which assumes that each review instance belongs to **exactly one single** cluster (group). However, this assumption may become problematic for review instances (even at the sentence level) that exhibit multiple topics (groups) to different degrees. As a result, we adopt another category of techniques: *topic modeling* which assigns multiple topics to each review instance. For example, the review *"Just add emojis and more themes."* is modeled as a distribution over two topics (50% "emoji", 50% "theme"). We will discuss the comparison of two algorithms in *topic modeling*, i.e., Latent Dirichlet Allocation (LDA) [11] and Aspect and Sentiment Unification Model (ASUM) [30] (adopted in [22]) in our experiments. In the future, we will explore and compare more topic models.

4.5 Ranking

Given the grouping results, we aim to determine the relative importance of (i) groups; and (ii) review instances in each group. To fulfill this purpose, we propose a novel ranking model presented as follows.

4.5.1 The Overview of Our Ranking Model

The general form of our ranking model is shown in Algorithm 1. The inputs include (i) a set of groups (topics) $\mathcal{G}$ generated by the grouping step; (ii) two sets of functions $\mathbf{f}^G$ (see Section 4.5.2) and $\mathbf{f}^I$ (see Section 4.5.3) that measure the importance of various features of groups (e.g., volume) and review instances (e.g., rating), respectively; and (iii) two weight vectors $\mathbf{w}^G$ ($w_i^G \in [0,1]$, $\sum_{i=1}^m w_i^G = 1$) and $\mathbf{w}^I$ ($w_i^I \in [0,1]$, $\sum_{i=1}^n w_i^I = 1$) correspond to $\mathbf{f}^G$ and $\mathbf{f}^I$, respectively. Algorithm 1 computes (i) the $GroupScore(g) \in [0,1]$ for each group $g \in \mathcal{G}$ (Line 1-3), and (ii) the $InstanceScore(r) \in [0,1]$ for each review instance $r \in g$ (Line 4-6). Larger $GroupScore(g)$ and $InstanceScore(r)$ indicate higher importance. Finally, Algorithm 1 outputs the ranking results.

Our ranking model is flexible, since we can obtain ranking results from different angles by adjusting the weight vectors of $\mathbf{w}^G$ and $\mathbf{w}^I$ (See Section 5.5.2). We also claim that our ranking model is extensible, because it can easily incorporate more features (See Section 6 for discussion).

4.5.2 Group Ranking

To measure the importance of different groups, we use $\mathbf{f}^G = \{f^G_{Volume}, f^G_{TimeSeries}, f^G_{AvgRating}\}$ in this work. Next,

Algorithm 1: The Ranking Model

Input: A set of groups $\mathcal{G}$, feature function sets $\mathbf{f}^G = \{f_1^G, \ldots, f_m^G\}$ and $\mathbf{f}^I = \{f_1^I, \ldots, f_n^I\}$, weight vectors $\mathbf{w}^G = (w_1^G, \ldots, w_m^G)$ and $\mathbf{w}^I = (w_1^I, \ldots, w_n^I)$

1 **for** *each group* $g \in \mathcal{G}$ **do**
2 Compute $f_1^G(g)$, ..., $f_m^G(g)$
3 Set $GroupScore(g) = \sum_{i=1}^m (w_i^G \times f_i^G(g))$
4 **for** *each review instance* $r \in g$ **do**
5 Compute $f_1^I(r)$, ..., $f_n^I(r)$
6 Set $InstanceScore(r) = \sum_{j=1}^n (w_j^I \times f_j^I(r))$
7 **end**
8 **end**

Output: Groups in decreasing order of $GroupScore$; review instances in each group in decreasing order of $InstanceScore$

Table 3: Per-review Distribution over Groups

(a) Review-Group Matrix

	g_1	...	g_m
r_1	$p_{r_1 g_1}$	...	$p_{r_1 g_m}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
r_n	$p_{r_n g_1}$	...	$p_{r_n g_m}$

(b) An Example

	g_1	g_2	g_3
r_1	1.0	0.0	0.0
r_2	1.0	0.0	0.0
r_3	0.5	0.5	0.0
r_4	0.0	0.5	0.5

we describe each feature function in detail.

Volume: Given the remaining n review instances (after filtering) $\mathcal{R} = \{r_1, \ldots, r_n\}$, in the grouping phase, we automatically discover m groups (topics), denoted as $\mathcal{G} = \{g_1, \ldots, g_m\}$. As described in Section 4.4, each review instance r_i ($1 \le i \le n$) is modeled as a distribution over $\mathcal{G}$. The matrix shown in Table 3(a) presents such distributions, where each entry $p_{r_i g_j}$ ($1 \le i \le n$,$1 \le j \le m$) represents the proportion that review instance r_i exhibits group g_j, and for each r_i, $\sum_{j=1}^m p_{r_i g_j} = 1$. For example, in Table 3(b), r_4 exhibits g_2 with 50% and g_3 with 50%. The volume of a group g is defined as follows,

$$f^G_{Volume}(g) = \sum_{i=1}^{n} p_{r_i g} \tag{1}$$

For example, in Table 3(b), $f^G_{Volume}(g_1) = 1+1+0.5+0 = 2.5$. One group with larger volume indicates it is more important. The reason is that a larger group is more likely to be a class of common bugs/requests reflected by many users, while a smaller group is more likely to be (i) a kind of particular bug/request reported by only a few users or (ii) a few users' wrong/careless operations.

Time Series Pattern: Given the time interval $\mathcal{T} = [t_0, t_0 + T]$ under investigation, we divide $\mathcal{T}$ into $K = T/\Delta t$ consecutive time windows, with each has length of Δt. Let $\mathcal{T}_k$ denote the k-th time window, thus $\mathcal{T}_k = [t_0 + (k-1)\Delta t, t_0 + k\Delta t]$, where $1 \le k \le K$. For each $r_i \in \mathcal{R}$, we denote $r_i.TS$ as the timestamp when r_i is posted, $t_0 \le r_i.TS \le t_0 + T$ for all $1 \le i \le n$. Given a time window $\mathcal{T}_k$, we denote the total number of review instances posted during it as follows,

$$v(\mathcal{T}_k) = |\mathcal{R}_{\mathcal{T}_k}| = |\{r_i : r_i.TS \in \mathcal{T}_k\}|, \qquad n = \sum_{k=1}^{K} v(\mathcal{T}_k)$$

where $|M|$ denotes the cardinality of the set M. For a group g, we count the volume of review instances posted during the time window $\mathcal{T}_k$, formally,

$$v(g, \mathcal{T}_k) = \sum_{r_j \in \mathcal{R}_{\mathcal{T}_k}} p_{r_j g}$$

Then, we can construct a time series for the group g, represented by,

$$P_g(\mathcal{T}, \Delta t) = [p(g,1), \ldots\ldots, p(g,K)]_g$$

where $p(g,k)$ is short for $p(g, \mathcal{T}_k)$, and $p(g,k) = v(g, \mathcal{T}_k)/v(\mathcal{T}_k)$.

Figure 4 shows four typical time series patterns. The pattern P_1 shown in Figure 4(a) has a rapid rise at a certain time window ($\mathcal{T}_k$) followed by a small decline then towards plateau. One group that has this kind of pattern is likely to be a class of newly introduced bug/request due to some event happened at $\mathcal{T}_k$ (e.g., version update, network/server error, and etc.). In addition, this problem is not solved at the end of $\mathcal{T}$. P_2 shown in Figure 4(b) presents a quick decay at a certain time window ($\mathcal{T}_k$). This demonstrates the scenario where an old bug/request is fixed/satisfied at $\mathcal{T}_k$. P_3 shown in Figure 4(c) fluctuates slightly within a range over the entire $\mathcal{T}$. This indicates the scenario of an existing bug/request introduced earlier than t_0 that is not fixed/satisfied during $\mathcal{T}$. P_4 shown in Figure 4(d) implies that the problem is introduced earlier than t_0 that is relieved (but not addressed) during $\mathcal{T}$. Obviously, groups with pattern P_1 are the most important (fresher), while groups of pattern P_2 are the least important (older). To model the importance of time series pattern for a group g, we compute $f^G_{TimeSeries}(g)$ by,

$$f^G_{TimeSeries}(g) = \sum_{k=1}^{K} \frac{p(g,k)}{p(g)} \times l(k) \tag{2}$$

where $p(g) = \sum_{k=1}^{K} p(g,k)$, and $l(k)$ is a monotonically increasing function of k (the index of $\mathcal{T}_k$), since we aim to set a higher weight to later $\mathcal{T}_k$. The choice of $l(k)$ depends on the importance of the freshness, in this work, we simply set $l(k) = k$.

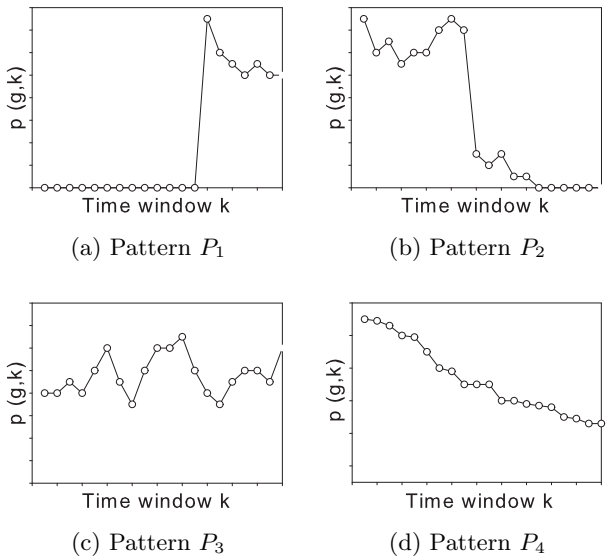
Average Rating: We denote $r_i.R$ as the user rating of r_i, in this work, $r_i.R \in \{1,2,3,4,5\}$ for all $1 \le i \le n$. "Informative" reviews with lower ratings (e.g., 1, 2) tend to express users' strong dissatisfaction with certain aspects of apps that need to be addressed immediately (e.g., critical bugs), thus are more important. On the other hand, "informative" reviews with higher ratings (e.g., 4, 5) often describe some kind of users' non-urgent requirements (e.g., feature improvement), thus are less important. Therefore, we measure the rating aspect of a group g by

$$f^G_{AvgRating}(g) = \frac{f^G_{Volume}(g)}{\sum_{i=1}^{n} p_{r_i g} \times r_i.R} \tag{3}$$

which is the inverted average rating of a group g. Larger $f^G_{AvgRating}(g)$ indicates more importance.

4.5.3 Instance Ranking

Regarding the importance of review instances in a particular group g, in this work, we use $\mathbf{f}^I = \{f^I_{Proportion}, f^I_{Duplicates}, f^I_{Probability}, f^I_{Rating}, f^I_{Timestamp}\}$. Next, we describe each feature function in detail.



(a) Pattern P_1 (b) Pattern P_2

(c) Pattern P_3 (d) Pattern P_4

Figure 4: Representative Time Series Patterns for Groups

Proportion: For a group g, each $r_i \in \mathcal{R}$ $(1 \le i \le n)$ exhibit g with a certain proportion $p_{r_i g}$. $p_{r_i g}$ equals to 1 means r_i only exhibits g, while $p_{r_i g}$ equals to 0 indicates r_i does not exhibit g at all. r_i with larger $p_{r_i g}$ value is more important in g, since it contains more core content of this group. Formally,

$$f^I_{Proportion}(r, g) = p_{rg} \tag{4}$$

In this work, we denote $\mathcal{R}_g$ as the reviews instances belong to a group g. $\mathcal{R}_g$ is constructed by eliminating those r_i with $p_{r_i g} < \alpha$ in $\mathcal{R}$ (we set $\alpha = 0.01$), thus $\mathcal{R}_g = \{r_i : p_{r_i g} \ge \alpha\}$. For example, in Table 3(b), $\mathcal{R}_{g_1} = \{r_1, r_2, r_3\}$, r_4 is ignored since $p_{r_4 g_1} = 0.0 < 0.01$.

Duplicates: For a group g of $\mathcal{R}_g = \{r_1, \ldots, r_{n_g}\}$, we denote $r.Text$ as the text of r (represented in the vector space). It is common that different texts of review instances refer to the same "informative" information. We intend to remove those duplicates from $\mathcal{R}_g$ and form a set of unique review instances $\mathcal{R}^u_g = \{r^u_1, \ldots, r^u_{n'_g}\}$, where $n'_g \le n_g$. Specifically, for each unique review instance $r^u_i \in \mathcal{R}^u_g$, $r_j \in \mathcal{R}_g$ is considered as a duplicate of r^u_i if and only if satisfying $\beta \le sim(r_j.Text, r^u_i.Text)$, where sim is a certain similarity metric (e.g., Jaccard similarity used in our work), and β is a predefined threshold. We count the number of duplicates for each $r^u_i \in \mathcal{R}^u_g$, denoted as $duplicates(r^u_i, g)$. The more duplicates r^u_i has, the more important it is in g. Formally,

$$f^I_{Duplicates}(r, g) = duplicates(r, g) \tag{5}$$

where r is a shorthand for r^u_i.

Note that, for a group g, we quantify the importance of every unique review instance $r \in \mathcal{R}^u_g$. For r that has more than one duplicate, the *rating* of r is set as the minimum rating value of duplicates, and the proportion, *probability* and *timestamp* of r are set as the maximum values of duplicates. The features in italics will be introduced below shortly.

Probability: As mentioned in Section 4.3, one of the reasons we choose EMNB as our classifier is that it can provide

a posterior probability for each predicated review instance (denoted as $r.P$). Intuitively, the larger probability of r demonstrates that it is more likely to be an "informative" review, thus more important. Formally,

$$f^I_{Probability}(r) = r.P \quad (6)$$

Rating: Similar to the average rating of a group, lower rating of r indicates it is more important, thus,

$$f^I_{Rating}(r) = \frac{1}{r.R} \quad (7)$$

Timestamp: Similar to the time series pattern of a group, more fresher of r indicates it is more important, thus,

$$f^I_{Timestamp}(r) = k \quad (8)$$

where k is the index of $\mathcal{T}_k$ that satisfies $r.TS \in \mathcal{T}_k$.

4.6 Visualization

The last step of AR-Miner is to visualize the results generated by our ranking model. Figure 5 shows a possible visualization of top-10 ranked results (see details in Section 5.4). The bottom half of Figure 5 is a radar chart which depicts the *GroupScore* value of each group (topic) along a separate axis which starts in the center of the chart and ends on the outer ring. Each group is labeled by two (or three) top probability (also descriptive) words within the group, and a data point near the outer ring indicates a higher *GroupScore* value. Intuitively, the group "more theme", which is requesting for more themes into the app, has the highest *GroupScore*, and the *GroupScore*s of the remaining groups decrease in the clockwise direction. To get insight into a group, we can click its label to view the reviews instances (the detailed information) within the group in decreasing order of *InstanceScore*. For example, the top half of Figure 5 shows the top 2 review instances in the "more theme" group. Due to space limitation, we present the complete version of Figure 5 on our website.

5. EMPIRICAL EVALUATION

To evaluate if AR-Miner can really help app developers, we conduct several experiments and case studies. Specifically, we aim to answer the following questions: (1) What is the topic discovering performance of our scheme? (2) If the top-ranked topics generated by AR-Miner represent the most "informative" user feedback for **real app developers**? (3) What are the advantages of AR-Miner over purely manual inspection and facilities used in traditional channels (e.g., online forum)?

5.1 Dataset

We select 4 Android apps from Google Play, i.e., SwiftKey Keyboard (smart touchscreen keyboard), Facebook (social app), Temple Run 2 (parkour game), and Tap Fish (casual game), as subject apps in our experiments and case studies. These apps are selected because (i) they cover different app domains; and (ii) they range from large datasets (Facebook and Temple Run 2) to relatively small datasets (SwiftKey Keyboard and Tap Fish). We collected the raw user reviews of these apps from Google Play roughly in the period from Oct, 2012 to Feb, 2013. Table 4 lists some key features of these datasets (after converting to sentence level). For each dataset, we divide it into two partitions, where reviews in partition (i) appear before those of partition (ii)

	Review Instances of topic "more theme"	Score
1	Also we need more themes!	0.932
2	Just wish you had more themes or ability to make a custom theme.	0.800
...	.....	.....

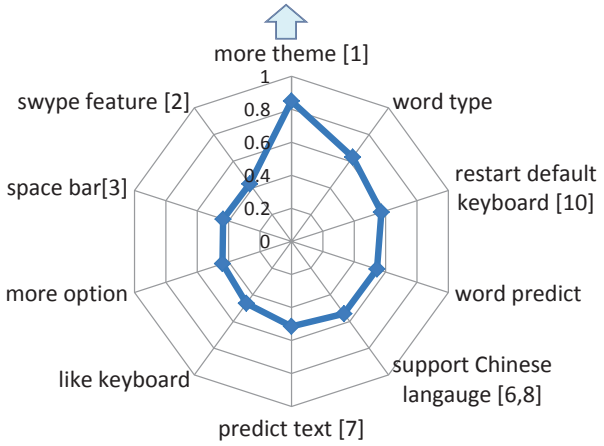


Figure 5: Visualization of Top-10 ranked results achieved by AR-Miner (SwiftKey). The number in the square bracket denotes the corresponding rank in Figure 6.

in terms of their posting time. We adopt some data from partition (i) for training, and some data from partition (ii) for test. Specifically, for partition (i), we randomly sample 1000 reviews as labeled training pool, and treat the rest as unlabeled data. For partition (ii), we randomly sample 2000 reviews for labeling and use as test set for evaluation.

Table 4: Statistics of Our Datasets (Train = Training Pool, Unlab. = Unlabeled Set).

Dataset	*SwiftKey*	*Facebook*	*TempleRun2*	*TapFish*
Train	1000	1000	1000	1000
Unlab.	3282	104709	57559	3547
Test	2000	2000	2000	2000
% Info.	29.3%	55.4%	31.1%	24.6%

We collected the ground truth labels of the training pool and test set according to the rules summarized in Figure 1. Each review is labeled by three different evaluators (some are workers in Amazon Mechanical Turk [1]), and the final label is determined by the "majority voting". The row "% Info." in Table 4 shows the proportion of "informative" reviews (among data with ground truth). On average, 35.1% reviews are "informative". Without loss of generality, we take "informative" as positive class and "non-informative" as negative class.

5.2 Performance Metrics

In this section, we introduce the performance metrics used in our evaluation. The first set of metrics include Precision, Recall (Hit-rate) and F-measure, which are defined below:

$$\text{Precision} = \frac{TP}{TP+FP}, \qquad \text{Recall(Hit}-\text{rate)} = \frac{TP}{TP+FN}$$

$$\text{F} - \text{measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

where TP, FP, FN represent the numbers of true positives (hits), false positives, and false negatives (misses), respectively. In addition, we also adopt the well-known Normalized Discounted Cumulative Gain (NDCG) [17] as a measure for evaluating the quality of top-k ranking results:

$$\text{NDCG@k} = \frac{DCG@k}{IDCG@k}$$

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

5.3 Evaluation of Grouping Performance

We conduct two experiments to evaluate the performance of our scheme (the first 3 steps shown in Figure 2) for automatically discovering groups (topics). First, we qualitatively compare our scheme (which contains a filtering step before grouping) with the baseline scheme used in [22] (which directly applies topic models). Second, we explore and compare two different settings of our scheme, i.e., (i) EMNB-LDA (Stanford Topic Modeling Toolbox [5] implementation for LDA); and (ii) EMNB-ASUM (the original implementation [2] with default parameters for ASUM), where ASUM is proposed in [30] and adopted in [22].

We select the EMNB filter for each dataset shown in Table 4 as follows. For each experimental trial, we randomly choose a subset of training pool (128 examples per class) as training data, and then apply the EMNB algorithm (the LingPipe implementation [4]) to build a classifier on the combination of training data and unlabeled set, finally measure the performance on the test set. We repeat the above experimental trial 50 times and choose the classifier with the best F-measure as the filter. Table 5 shows the F-measure attained by the four selected filters used in our experiments. We can see that, their performance is fairly good, especially the *Facebook* filter (0.877).

Table 5: The Performance of Selected Filters

Filter	*SwiftKey*	*Facebook*	*TempleRun2*	*TapFish*
F-measure	0.764	0.877	0.797	0.761

5.3.1 Qualitative Comparison of Both Schemes

The first experiment qualitatively compares our scheme (EMNB-LDA) with the baseline scheme (LDA). We apply both schemes to the test set of each dataset shown in Table 4 after preprocessing. We vary the number of topics (denoted as K) and choose the appropriate K values according to (i) the *perplexity* scores [11] on 20% held-out data (should be small); and (ii) the results themselves (should be reasonable). Table 6 shows some representative topics found by EMNB-LDA and LDA from the test set of *SwiftKey*. For each topic, we list the top-10 weighted words in the vocabulary distribution. For space reasons, we do not present the results for other datasets (which are similar).

From the results shown in Table 6, we can draw two observations. First, most topics found by EMNB-LDA are "informative", e.g., "theme", "Chinese", "jelly bean", "predict" and "space" shown in Table 6(a), while LDA presents many "non-informative" (or redundant) topics, such as "type"(purely praise without any advice) and "worth" (emotional expression) shown in Table 6(b) in red color. The reason is straightforward: LDA does not have a filtering phase. Second, although with well-tuned K value, LDA could also find "informative" topics discovered by EMNB-LDA, some of them have lower quality. For example, the topic "jelly bean" shown in Table 6(b) has (i) lower-ranked key words; and (ii) lower purity (the word "predict" ranked high).

Table 6: Some topics found by EMNB-LDA (K=20) and LDA (K=36) on "SwiftKey" dataset. The colored words are topic labels.

(a) EMNB-LDA

theme	Chinese	jelly bean	predict	space
more	languag	bean	word	space
theme	chines	jelli	predict	period
wish	need	galaxi	text	email
love	wait	note	complet	enter
custom	user	keyboard	auto	insert
like	download	samsung	like	automat
color	support	screen	pen	input
star	input	updat	won	mark
option	except	android	basic	address
keyboard	thai	swiftkei	automat	dont

(b) LDA

theme	Chinese	jelly bean	type	worth
theme	chines	**predict**	type	worth
more	languag	text	make	monei
like	faster	**bean**	easi	definit
color	input	**jelli**	learn	paid
love	more	time	predict	penni
wish	need	issu	easier	price
custom	switch	accur	speed	download
option	annoi	start	accur	total
pick	time	browser	perfectli	cent
red	write	samsung	time	amaz

In sum, we can conclude that our scheme (with filtering) performs better than the baseline scheme (without filtering) in solving our problem.

5.3.2 Comparison of Two Topic Models

The second experiment is to compare the performance of two topic models (LDA and ASUM) in our scheme. For each dataset shown in Table 4, we manually identify one appropriate and representative group from "informative" reviews in the test set as ground truth (prior to running our scheme), where each review in the group is assigned a proportion score. The "Topic" column of Table 7 shows the labels of the groups. Following the same setup as the first experiment (K=20), we evaluate the performance of EMNB-LDA and EMNB-ASUM by measuring if they can discover the pre-identified groups accurately. Table 7 presents the experimental results in terms of F-measure (averaged over 50 iterations).

Some observations can be drawn from the results shown in Table 7. First, for all topics, EMNB-LDA performs better than EMNB-ASUM in terms of F-measure. One possible reason is ASUM imposes a constraint that all words in a sentence be generated from one topic [30, page 2]. Thus, sentence-level reviews exhibit several topics are only assigned to one topic, which results in information lost. Second, by looking into the results, the F-measure achieved by

Table 7: Evaluation Results, K = 20

Dataset	Topic	EMNB-ASUM	EMNB-LDA
SwiftKey	"theme"	0.437	**0.657**
Facebook	"status"	0.388	**0.583**
TempleRun2	"lag"	0.210	**0.418**
TapFish	"easier buck"	0.386	**0.477**

EMNB-LDA is reasonable but not promising, e.g., 0.657 for the "theme" topic of *SwiftKey*. The main reason is the unsupervised topic modeling is a hard task. Besides, some "informative" reviews are removed wrongly by the filter, while some "non-informative" ones are not filtered out.

5.4 Evaluation of Ranking Performance

In this section, we report a comprehensive case study to evaluate the ranking performance of AR-Miner. We aim to examine whether the top-ranked topics generated by AR-Miner represent the most "informative" user feedback for **real app developers**.

We use SwiftKey Keyboard shown in Table 4 as our subject app. The developers of this app created a nice *SwiftKey feedback forum* to collect user feedback [6]. It provides users a voting mechanism for every feedback, and feedback with high-voting is ranked top. Feedback considered to be "informative" to developers is assigned a *Status*, which shows the current progress of this feedback. Therefore, we can obtain a comparable ranking list of "informative" information for **real** developers of SwiftKey Keyboard. Specifically, we first selected all the user feedback in the forums, and then removed those feedback without *Status* (indicates "non-informative" to developers) or assigned *Status* of "Complete" (indicates closed) before the time interval $\mathcal{T}$ (from Oct 12th, 2012 to Dec 19th, 2012) we investigated, finally we ranked the remaining feedback in the decreasing order of number of votes. The top-10 ranked results (ground truth) are shown in Figure 6 (verified around Feb 17th, 2013).

The user reviews of SwiftKey Keyboard collected in $\mathcal{T}$ contains 6463 instances. We use the filter shown in Table 5, and apply both LDA and ASUM algorithms. Finally, the top-10 ranked results generated by our ranking model are visualized in Figure 5 (LDA setting, K = 22, $\beta = 0.6$, $\mathbf{w}^G = (0.85, 0, 0.15)$, $\mathbf{w}^I = (0.2, 0.2, 0.2, 0.2, 0.2)$). We compare the ranking results attained by AR-Miner with the ground truth ranking results (Figure 6), and measure the comparison in terms of Hit-rate and NDCG@10 scores introduced in Section 5.2. Note that, each feedback shown in Figure 6 is considered to be corresponding to a topic shown in Figure 5 if and only if (i) the feedback is closely related to the topic, and (ii) the (semantically similar) feedback can be found in the top review instances of the topic. Table 8 presents the results.

Remark. We use the ranking list shown in Figure 6 as the ground truth mainly because it is the choice of **real** app developers in real scenarios. We believe it is much more convincing than a ranking list identified by others (e.g., the authors). Besides, we assume that the greater agreement between the ranking results achieved by AR-Miner and the ground truth, the better performance of AR-Miner. Specifically, if both Hit-rate and NDCG@10 equal to 1, we think that AR-Miner is as effective as *SwiftKey feedback forum*.

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

Figure 6: Top-10 Ranked Results Attained from SwiftKey Feedback Forum, Highlighted Feedback Has Corresponding Topic Shown in Figure 5

Table 8: Ranking Results

	AR-Miner (LDA)	AR-Miner (ASUM)
Hit-rate	**0.70**	0.50
NDCG@10	**0.552**	0.437

Some observations can be found from Table 8. First, for both metrics, LDA performs better than ASUM in our framework. Second, by looking into the results, AR-Miner (LDA) achieves 0.70 in terms of Hit-rate, which indicates that AR-Miner is able to automatically discover the most "informative" information effectively, and thus can be beneficial for app developers, especially those who have not established valid channels. Feedback highlighted in Figure 6 has corresponding topic in the radar chart shown in Figure 5. For example, the ranked **1st** topic discovered by AR-Miner shown in Figure 5 ("more theme [1]", where the number in square bracket represents the rank in Figure 6) corresponds to the ranked **1st** user feedback ("More themes. More themes. More themes.") shown in Figure 6.

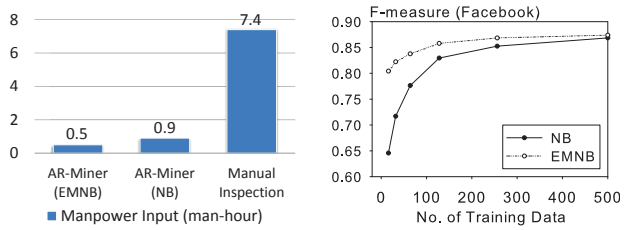
5.5 Comparison with Manual Inspection and Traditional Channels

We conduct two case studies to (i) compare AR-Miner with manual inspection in terms of manpower input; and (ii) analyze the advantages of AR-Miner over facilities used in a traditional channel (i.e., online forum).

5.5.1 Manpower Input Analysis

In the first case study, we apply three schemes: (i) AR-Miner with EMNB filter (256 training examples); (ii) AR-Miner with NB filter (500 training examples); and (iii) purely manual inspection, respectively, to the test set of the *Facebook* dataset shown in Table 4 (2000 examples). We recorded the approximate manpower input (of the first author) for finding the most "informative" information by these three schemes[2]. For simplicity, we ignore the performance difference between AR-Miner and manual inspection. Figure 7(a) presents the comparison results.

[2]For purely manual inspection, we recorded the efforts spent on sampled data, and then estimated the total man-hours.



(a) Manpower Comparison (b) EMNB vs. NB

Figure 7: Evaluation Results on "Facebook". (a) manpower comparison with manual inspection, (b) comparison between EMNB and NB with varied training data.

Some observations can be found from the results shown in Figure 7(a). First, we find that AR-Miner (EMNB filter, 0.5 man-hours) is much more efficient than purely manual inspection (7.4 man-hours). The reason is AR-Miner only requires humans to label some training data, and can work automatically after the filter has been built. Second, AR-Miner (NB) needs more human efforts than AR-Miner (EMNB), since building a NB filter whose performance is comparable to a EMNB filter requires manually labeling more training data (500 and 256 examples for NB and EMNB, respectively, in this case study). We explain it with the results shown in Figure 7(b). Following the same setup described in paragraph 2 of Section 5.3, Figure 7(b) shows the average F-measure of NB and EMNB under varying amounts of training data (*Facebook*). It is obvious that, when the F-measure score is fixed, NB always requires more training data (human efforts) than EMNB (the results are similar for other datasets, check details on our website). Therefore, we choose EMNB in AR-Miner to reduce human efforts as much as possible.

5.5.2 Comparison with an Online Forum

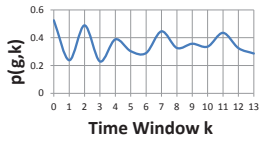
Following the same setup of SwiftKey Keyboard described in paragraph 3 of Section 5.4, we conduct a case study to analyze the advantages of AR-Miner over a traditional channel, i.e., online forum (*SwiftKey feedback forum*).

First of all, from user reviews, AR-Miner has the ability to discover fresh "informative" information that does not exist in the *SwiftKey feedback forum*. Take the ranked 1st topic "more theme" shown in Figure 5 as an example. Figure 8(a) shows more review instances in the top-10 list of "more theme". The top 1 ranked review shown in Figure 8(a) and the ranked 1st user feedback shown in Figure 6 are semantically the same. Moreover, we observe that the ranked 10th review ("*...., or support for third party themes*") is only discovered by AR-Miner, which offers app developers new suggestions concerning the topic "more theme". This kind of new information is beneficial to developers, since it may inspire them to further improve their apps.

Second, AR-Miner can provide app developers deep and more insights than *SwiftKey feedback forum* by flexibly adjusting the weight vectors of $\mathbf{w}^G$ and $\mathbf{w}^I$. For example, as described in Section 5.4, *SwiftKey feedback forum* only support a user voting mechanism (like *Volume* in $\mathbf{w}^G$) to rank the user feedback, while AR-Miner can achieve it from different angles. If setting $\mathbf{w}^G = (0.0, 0.0, 1.0)$ (indicates groups are ranked only according to *AvgRating*), the ranking of "more theme" shown in Figure 5 drops from 1 to 22, which implies that it's not a kind of critical and urgent problem

	Topic: "more theme"	Score
1	Also we need more themes!	0.932
2	Just wish you had more themes or ability to make a custom theme.	0.800
...	.....	.....
10	It would also be nice to have more themes or support for third party themes.	0.621

(a) Instances Ranking



(b) P of "theme"

Figure 8: Unique Information Offered by AR-Miner

to users. If setting $\mathbf{w}^G = (0.0, 1.0, 0.0)$ (indicates groups are ranked only according to *TimeSeries*), the ranking of "more theme" drops from 1 to 18. The time series pattern of "more theme" in this case can be automatically visualized as shown in Figure 8(b), which helps app developers easily understand that it's a kind of existing problem.

In sum, this case study implies that even for those app developers who have already established some traditional channels, AR-Miner can be a beneficial compliment.

6. LIMITATIONS AND THREATS TO VALIDITY

Despite the encouraging results, this work has two potential threats to validity. First, the authors are not professional app developers, and thus the defined category rules of informativeness as summarized in Figure 1 might not be always true for real app developers. In this paper, we have attempted to alleviate this threat by (i) studying what kinds of user feedback are *real* app developers concerned with; and (ii) exploiting *real* app developers' decisions as the ground truth for evaluation. The second threat relates to the generality of our framework. We validate our framework on user reviews of four Android apps from Google Play. It is unclear that if our framework can attain similar good results when being applied to other kinds of Android apps (e.g., apps in Amazon Appstore) and apps on other platforms (e.g., iOS). Future work will conduct a large-scale empirical study to address the threat. Besides, another limitation of our work is that we only choose $\mathcal{A} = \{Text, Rating, Timestamp\}$ as mentioned in Section 3, but a real app marketplace may have more features of user reviews (e.g., *Device Name* in Google Play, *Amazon Verified Purchase* in Amazon Appstore). The impact of these specific features is unknown, but our framework is rather generic and extensible to incorporating more features in future work.

7. CONCLUSION

This paper presented AR-Miner, a novel framework for mobile app review mining to facilitate app developers extract the most "informative" information from raw user reviews in app marketplace with minimal manual effort. We found encouraging results from our extensive experiments and case studies, which not only validates the efficacy but also shows the potential application prospect of AR-Miner. We also discuss some limitations along with threats to validity in this work, and plan to address them in the future.

8. ACKNOWLEDGMENTS

This work was in part supported by the Singapore MOE tier-1 research grant (RG33/11). Special thanks to Shaohua Li for helping us label some of the data. We also thank the anonymous reviewers for their greatly helpful comments.

9. REFERENCES

[1] Amazon Mechanical Turk. https://www.mturk.com/.

[2] Aspect and Sentiment Unification Model. http://uilab.kaist.ac.kr/research/WSDM11/. [accessed 21-Jan-2014].

[3] Bugzilla. http://www.bugzilla.org/.

[4] LingPipe. http://alias-i.com/lingpipe/.

[5] Stanford Topic Modeling Toolbox. http://nlp.stanford.edu/software/tmt/tmt-0.4/.

[6] SwiftKey Feedback Forums. http://support.swiftkey.net/. [accessed 21-Jan-2014].

[7] M. Abulaish, Jahiruddin, M. N. Doja, and T. Ahmad. Feature and opinion mining for customer review summarization. In *Proceedings of the 3rd International Conference on Pattern Recognition and Machine Intelligence*, pages 219–224, 2009.

[8] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, pages 23:304–23:318, 2008.

[9] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, pages 361–370, 2006.

[10] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza. Content classification of development emails. In *Proceedings of the 34th International Conference on Software Engineering*, pages 375–385, 2012.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.

[12] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, 2006.

[13] R. Chandy and H. Gu. Identifying spam in the ios app store. In *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality*, pages 56–59, 2012.

[14] P. H. Chia, Y. Yamamoto, and N. Asokan. Is this app safe?: A large scale study on application permissions and risk signals. In *Proceedings of the 21st International Conference on World Wide Web*, pages 311–320, 2012.

[15] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 155–164, 2010.

[16] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *Proceedings of the 14th International Requirements Engineering Conference*, pages 36–45, 2006.

[17] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice.* Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[18] O. Dalal, S. H. Sengemedu, and S. Sanyal. Multi-objective ranking of comments on web. In *Proceedings of the 21st International Conference on World Wide Web*, pages 419–428, 2012.

[19] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel. Rebucket: A method for clustering duplicate crash reports based on call stack similarity. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1084–1093, 2012.

[20] X. Ding, B. Liu, and P. S. Yu. A holistic lexicon-based approach to opinion mining. In *Proceedings of the 1st International Conference on Web Search and Data Mining*, pages 231–240, 2008.

[21] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1276–1284, 2013.

[22] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 35th International Conference on Software Engineering*, pages 582–591, 2013.

[23] H. georg Kemper and E. Wolf. Iterative process models for mobile application systems: A framework. In *Proceedings of the 23rd International Conference on Information Systems*, pages 401–413, 2002.

[24] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: Msr for app stores. In *Proceedings of the 9th Working Conference on Mining Software Repositories*, pages 108–111, 2012.

[25] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 35th International Conference on Software Engineering*, pages 392–401, 2013.

[26] C.-F. Hsu, E. Khabiri, and J. Caverlee. Ranking comments on the social web. In *Proceedings of the 2009 International Conference on Computational Science and Engineering*, pages 90–97, 2009.

[27] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 168–177, 2004.

[28] 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*, pages 41–44, 2013.

[29] N. Jindal and B. Liu. Opinion spam and analysis. In *Proceedings of the 1st International Conference on Web Search and Data Mining*, pages 219–230, 2008.

[30] Y. Jo and A. H. Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the 4th International Conference on Web Search and Data Mining*, pages 815–824, 2011.

[31] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, pages 477–487, 2013.

[32] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[33] R. Minelli and M. Lanza. Software analytics for mobile applications–insights & lessons learned. In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, pages 144–153, 2013.

[34] A. Mukherjee, B. Liu, and N. Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st International Conference on World Wide Web*, pages 191–200, 2012.

[35] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Mach. Learn.*, 39(2-3):103–134, May 2000.

[36] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *Proceedings of the 21st IEEE International Requirements Engineering Conference*, pages 125–134, 2013.

[37] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, Jan. 2008.

[38] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 45–54, 2010.

[39] B. Turhan and A. B. Bener. Software defect prediction: Heuristics for weighted naïve bayes. In *Proceedings of the 2nd International Conference on Software and Data Technologies, Volume SE*, pages 244–249, 2007.

[40] P. D. Turney. Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 417–424, 2002.

[41] S. Xie, G. Wang, S. Lin, and P. S. Yu. Review spam detection via temporal pattern discovery. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 823–831, 2012.

[42] L. Zhuang, F. Jing, and X.-Y. Zhu. Movie review mining and summarization. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 43–50, 2006.