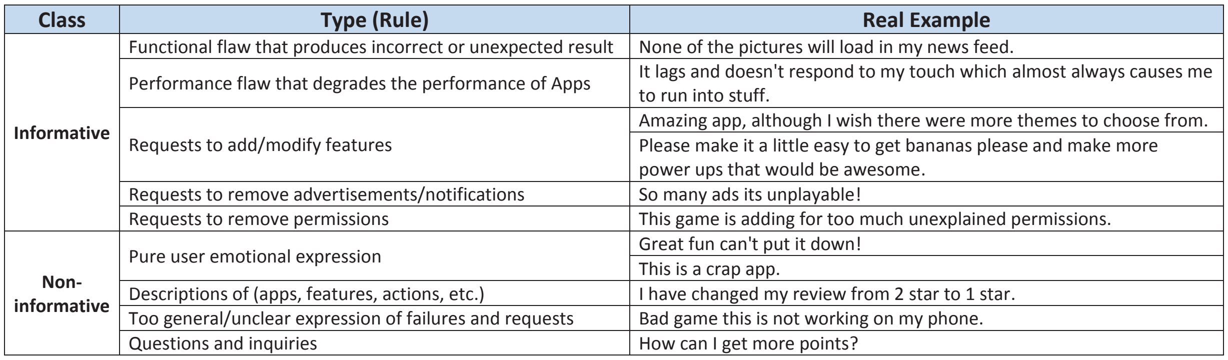

Class	Type (Rule)	Real Example
Informative	Functional flaw that produces incorrect or unexpected result	None of the pictures will load in my news feed.
	Performance flaw that degrades the performance of Apps	It lags and doesn't respond to my touch which almost always causes me to run into stuff.
	Requests 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)