2015 30th IEEE/ACM International Conference on Automated Software Engineering

“What parts of your apps are loved by users?[1]”

Xiaodong Gu and Sunghun Kim

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology, Hong Kong
{xguaa, hunkim}@cse.ust.hk

***Abstract*—Recently, Begel et al. found that one of the most important questions software developers ask is “what parts of software are used/loved by users.” User reviews provide an effective channel to address this question. However, most existing review summarization tools treat reviews as bags-of-words (i.e., mixed review categories) and are limited to extract software aspects and user preferences.**

We present a novel review summarization framework, SUR-Miner. Instead of a bags-of-words assumption, it classifies reviews into five categories and extracts aspects in sentences which include evaluation of aspect using a pattern-based parser. Then, SUR-Miner visualizes the summaries using two interactive diagrams. Our evaluation on 17 popular apps shows that SUR-Miner summarizes more accurate and clearer aspects than state-of-the-art techniques, with an average F1-score of 0.81, significantly greater than that of ReviewSpotlight (0.56) and Guzmans’ method (0.55). Feedback from developers shows that 88% developers agreed with the usefulness of the summaries from SUR-Miner.

***Index Terms*—Review Summarization; User Feedback; Sentiment Analysis; Data Mining**

I. INTRODUCTION

Often software developers are eager to know what parts of their software is used/loved by users. According to a survey covering 4,000 Microsoft engineers, the question “*What parts (aspects) of a software product are most used and/or loved by customers?*” ranks the second among the top 145 questions developers asked [5]. This question requires developers to analyze preferences for and opinions toward different software aspects.

User reviews are an important channel for software developers to understand users’ requirements, preferences and complaints [21], [31]. Through analyzing user reviews, developers can evaluate their products, identify users’ preference [21], and improve software maintenance and evolution tasks [33].

Yet understanding software reviews is very challenging and tedious. First, the volume of user reviews is too large to be checked manually. Developers receive hundreds or thousands of reviews every day [10], [31]. Given the large number of reviews, they need to read and manually classify the reviews into complaints or new feature requests [30]. Such processes are extremely time-consuming and tedious. On the other hand, user reviews fall into too many varieties that need to be distinguished [31]. They can be new feature requests, bug reports, praises, or complaints. Different types of reviews target different tasks and developers [30]. For example, a praising review may not be valuable for software testing but can be essential for product evaluation. A review reporting a bug is not important for requirements analysis but can be crucial for software testing. Given millions of reviews, developers must first categorize them manually [30].

A few tools are proposed for software user review summarization. For example, Chen et al. [10] filter non-informative reviews by a classification technique and apply Latent Dirichlet Allocation (LDA) [6] to summarize topics of the informative reviews. Fu et al. [15] filter rating-inconsistent reviews, which have sentiments different from their rating by a regression model. They also apply LDA to summarize topics in the remaining reviews and show rating trends for different topics. Iacob et al. [21] filter reviews that request new features by linguistic rules and summarize key words of the requests with LDA. These tools summarize informative and reliable reviews. However, the LDA model that they used is based on a bag-of-word assumption without considering sentence structures and semantics. Such assumption may be problematic for software reviews which exhibit multiple purposes (e.g., aspect evaluation and feature request) and sentiments. Since these tools mix up aspects and opinions, and mix topics related to different categories, they are not effective to gauge users’ sentiments toward each aspect.

To address these limitations, we propose Software User Review Miner (SUR-Miner), a framework that can summarize users’ sentiments and opinions toward corresponding software aspects. Instead of treating reviews as bags of words, SUR-Miner makes full use of the monotonous structure and semantics of software user reviews, and directly parses aspect-opinion pairs from review sentences based on pre-defined sentence patterns. It then analyzes sentiments for each review sentence and associate sentiments with aspect-opinion pairs in the same sentence. Finally, it summarizes software aspects by clustering aspect-opinion pairs with the same aspects.

We empirically evaluate the performance of SUR-Miner on recent user reviews of 17 Android apps such as Swiftkey, Camera360, WeChat and Templerun2. We measure the performance of key processes (i.e., classification, aspect-opinion extraction and sentiment analysis) by F1-score which is a common accuracy measure in the text mining literature [14], [38]. Results show that the SUR-Miner produces reliable summaries, with average F1-scores of 0.75, 0.85 and 0.80 for review classification, aspect-opinion extraction and sentiment analysis, respectively. The final aspects from SUR-Miner are significantly more accurate and clearer than state-of-the-art techniques, with an F1-score of 0.81, greater than that of ReviewSpotlight (0.56) and Guzmans’ method (0.55).

[1]This question is from a study by Begel et al. at Microsoft [5]

As a proof-of-concept application, we design two interactive diagrams, aspect heat map and aspect trend map, using the summaries from SUR-Miner to help developers grasp users' preferences and typical opinions towards each software aspect. Feedback from corresponding app developers is also encouraging, with 88% of respondents agreeing that the summaries of SUR-Miner are useful, indicating that SUR-Miner helps developers understand users' preferences for different aspects in practice.

Overall, our study makes the following contributions:

1) We leverage a classification technique in which we designed text features to distinguish five review categories such as bug reports and new feature requests.
2) We propose a pattern-based parsing technique which can parse complex app review sentences, and extract aspects and corresponding opinions.
3) We design novel interactive visualizations to present summaries efficiently for app developers and managers.
4) We conduct an empirical evaluation of SUR-Miner to investigate its usefulness.

The rest of this paper is organized as follows. Section II presents the related work. Section III presents the detailed design of our framework. Section IV presents the evaluation. Section V discusses the threats to validation, and Section VI concludes the paper.

II. RELATED WORK

A. App Review Filtering

App review filtering has drawn increasing attention in the software engineering community. Chen et al. [10] filter non-informative reviews and rank the user review by significance. Their framework trains a classifier and categorizes reviews into two classes, namely, informative and non-informative. Fu et al. [15] filter rating-inconsistent reviews (reviews that have sentiments different from their rating) by a regression model on the review vocabulary. These tools can partially select informative reviews. However, they do not define clearly under which circumstances reviews are informative since different developers need different types of reviews [30], [31]. To a further step of their work, we aim at distinguishing different review purposes (categories) and selecting reviews from a specific category to extract and summarize software aspects.

Recent work by Sorbo et al. [13] proposes a similar idea to classify development emails according to their purposes. They also design a classification approach using natural language parsing techniques. While their technique could also be applied for app review classification, it does not support aspect summarization within each category.

B. Aspect Extraction from App Reviews

Aspect extraction also has been widely investigated in software engineering. Chen et al. [10] use LDA [6] to extract topics of reviews. Hu et al. [19] propose a method for web review mining. Their method extracts frequent words as aspects and link corresponding adjective words as opinions. Fu [15] address the problem of mining users' negative feedback. They apply LDA topic model to mine topics from negative feedback and rank the summarized problems for each release. Galvis et al. [16] mine requirement changes by adapting a topic model named Aspect and Sentiment Unification Model (ASUM) [22]. They also extract common topics and present users' opinions toward those topics.

However, their approaches differ significantly from ours. They applied frequent items mining or topic models which are based on a bag-of-word assumption without considering sentence structures and semantics. That means they can distinguish neither review categories (praising, feature requests, bug reports, shortcomings) nor aspects and user opinions, which could result in inaccuracy and confusion. For example, a topic word "prediction" extracted by LDA may mean that users appreciate the prediction feature or alternatively, users wish for a new prediction feature. In such case, developers cannot efficiently interpret the topics.

Recent work by Sarro et al. extracts features from app descriptions using natural language processing [34]. Our work differs to theirs in that we extract features from app reviews. Besides, we aim at summarizing app features, while their goal is to investigate feature lifecycles [34].

To our best knowledge, there is only one previous work that are closely related to ours. Guzman and Maalej [17] proposed to extract software features and analyze their sentiments. Our work differs from theirs in three major aspects. First, our approach aims not only to identify features, but also to distinguish feature evaluations and feature requests. Second, our approach can identify complex and novel features since it parses review sentences with semantic patterns, while their techniques are based on frequent item mining and topic model as traditional approaches did. Finally, we propose interactive visualizations to help app managers and developers grasp the feature evaluations and sentiment trends.

C. Review Mining in Other Marketplaces

User review mining is also an attractive topic in other marketplaces (e.g., commodity goods, movies). Yatani et al. [37] proposed a review summarization tool called ReviewSpotlight which extracts aspect-opinion pairs by identifying adjective-noun word pairs from review sentences. Huang et al. [20] adopted a similar idea and designed Revminer - an extractive interface for summarizing restaurant reviews. Nichols et al. [29] proposed ReCloud, which parses review sentences with NLP techniques. Zhuang et al. [38] studied movie review summarization. Their approach integrates multiple knowledge including WordNet, statistical analysis and movie knowledge.

However, these techniques can hardly be applied to app reviews directly. App reviews are quite different from reviews in other marketplaces [10], [15]. They have different lexicons and formats that existing tools can hardly parse. The ReviewSpotlight [37] presents a word cloud by extracting adjective-noun word pairs. Likewise, the RevMiner [20] extracts word pairs using a bootstrapping algorithm. The ReCloud [29] takes semantic into consideration, it also presents a word cloud but with a spatial layout reflecting the NLP context. However, app reviews cannot simply be represented by word clouds or word pairs. For example:

case 1: *"I love the fact that we can change themes"*

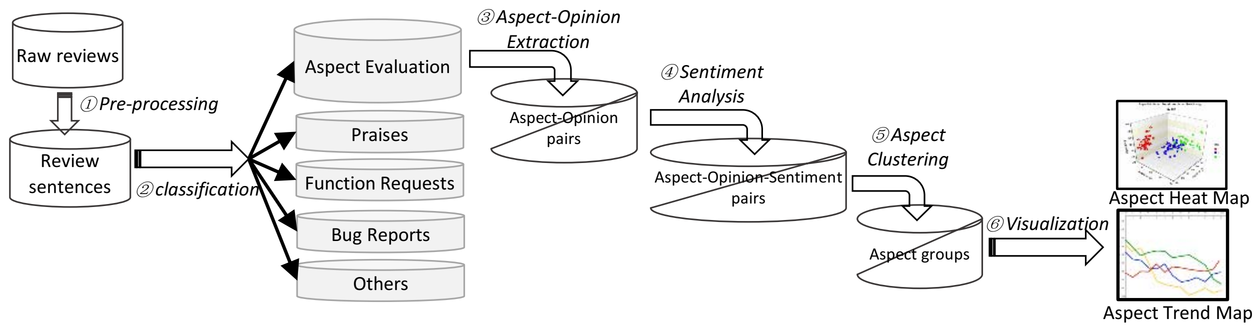


Fig. 1: Overview of the proposed SUR-Miner framework

The ReviewSpotlight cannot output anything since there is no adjective. The RevMiner and ReCloud may present some meaningless word pairs. In contrast, SUR-Miner can present the correct pair $\langle$*we can change themes, love*$\rangle$ as it considers semantics and app review patterns. In addition, app reviews contain multiple purposes that target different developers [31]. None of existing tools can distinguish such categories. Consider the following cases

case 2: *"The blue screen after clicking the 'ok' button is annoying."*

case 3: *"A simple UI would be better."*

From developers' perspective, they are just a bug report and a feature request and should not be considered as users' opinions toward "screen" and "UI". Such cases account for a large proportion in app reviews [31]. While all these tools still output word pairs(cloud) such as $\langle button, annoying\rangle$ and $\langle UI, simple\rangle$, SUR-Miner can distinguish the above cases as it leverages a classification technique.

III. SUR-MINER

This section introduces the generic architecture of SUR-Miner.

As illustrated in Figure 1, our framework takes user reviews including texts and ratings as inputs and outputs the main opinions and sentiments toward different aspects of the app. The whole procedure consists of six main steps: For raw reviews that need to be summarized, we first split them into sentences (step 1). Then, we classify each sentence into five categories, namely, *aspect evaluation*, *praises*, *feature requests*, *bug reports* and *others* (step 2). Then, we only select sentences in the *aspect evaluation* category and filter out other types of sentences. We then extract aspects and corresponding opinions and sentiments from the set of "aspect evaluation" sentences (step 3-4). The resulting aspect-opinion-sentiment pairs are clustered and visualized with two interactive diagrams (step 5-6). Each step is explained in detail herein below.

A. Step 1 - Preprocessing

The raw user review needs preprocessing. It often consists of more than one sentences with different purposes. For example, a raw review "*The UI is ugly. I want a beautiful UI*" consists of two sentences. The first sentence is an evaluation of an aspect *UI*, while the second is a request for improvement in the aspect *UI*. They have different purposes and sentiments. Therefore, it would be desirable to separate these sentences for analysis. Furthermore, user reviews have many typos and contractions which make it hard to understand the meaning automatically.

To address these two issues, we split the raw review text into sentences using the Stanford CoreNLP tool [26]. Each review sentence is time stamped and assigned rating, to be the same as in its raw review. We also correct common typos, contractions and repetitions such as "U→you", "coz→because", "&→and", "Plz→Please", "soooo→so" and "thx→thanks". We collected 60 such typos and contractions, and replaced them with regular expressions[2].

B. Step 2 - Review Classification

As discussed in Section I, review sentences may have different categories [31]. Different categories target different tasks and developers [30]. It is very tedious and time consuming for developers to manually classify them and select appropriate sentences for aspect evaluation. In the review classification step, we aim to automatically classify and select review sentences which contain aspect evaluation.

We define five review categories including *aspect evaluation*, *bug reports*, *feature requests*, *praise* and *others*. Pagano et al. found 17 categories (topics) of user reviews [31]. We use top four categories from their taxonomy [31] and merge other minor categories into an "others" category. Table I illustrates the definitions and sample review sentences for each category.

To classify review sentences into the above mentioned categories, we follow a supervised machine learning approach. We first collect historical review sentences, extract their text features, and manually label them according to the definitions in Table I. Then, we train a classifier using these text features and labels. Finally, we execute the classifier on new review instances to predict their categories.

We adopt a well-known classifier, *Max Entropy* which has great performance on text classification [24], [28]. In the following, we present the text features that we designed for classification.

1) Text Feature Extraction: We extracted two dimensions of text features: lexicon features and structural features.

Lexicons are important to characterize review categories since different review categories may have significantly different lexicons. For example, "amazing" and "great" appear

[2]The full list of typos is at http://www.cse.ust.hk/~xguaa/srminer/appendix.html

TABLE I: Definition of Five Review Categories [31]

Category	Definition	Examples
Praise	Expressing emotions without specific reasons	Excellent! I love it! Amazing!
Aspect Evaluation	Expressing opinions for specific aspects	The UI is convenient. I like the prediction text.
Bug Report	Reporting bugs, glitches or problems	It always force closes when I click the ".com" button.
Feature Request	Suggestions or new feature requests	It would be better if I could give opinion on it. It's a pity it doesn't support Chinese. I wish there was a "deny" button.
Others	Other categories that are defined in [31]	I've been playing it for three years

frequently in *praising reviews*, while "bug" and "fix" are representative words for *bug reports*. We choose *character N-Gram* and *trunk word* as two lexicon features since they reflect lexicons of different categories.

Character N-Gram Character N-Gram, an important lexical representation, is a commonly used feature in text classification [8], [9], [18], [23], [25]. It has also been found to be effective in many applications such as malicious code detection [4] and duplicate bug report detection [36] in software engineering. Character N-Gram features for a sentence are all n consecutive letters in the tokens of that sentence. For example, the 3-Grams for the sentence "The UI is OK" are *The, heU, eUI, UIi, Iis, isO,* and *sOK*. We use 2-4 Grams for classification.

Trunk Words We also propose *trunk word* as a lexicon feature. We define *trunk word* as the word at the root of a semantic dependence graph [12] which is introduced later in this section. For example, the trunk word of the sentence "The graphics are amazing" is "are".

Sentence structures can also reflect text features as different review categories may have different syntax and semantics. For example, for *aspect evaluations*, users tend to use descriptive syntax such as "*The graphic (noun) is amazing (adjective)*", while for *feature request*, users often use imperative sentences such as "*please add more themes*" and "*It could be better to have more themes (noun)*".

We leverage three structural features: POS tags, parsing tree, and semantic dependence graph.

POS tag Part Of Speech (POS) [11] is a widely used grammatical feature for texts. It indicates the property of each word in a sentence. For example, POS tags for sentence "The user interface is beautiful" are DT-NN-NN-VBZ-JJ in sequence [11]. Here, the POS tag for the word *is* is VBZ, which means *is* is a verb of 3rd person present singular. We generate POS tags using the Stanford CoreNLP tools [3], [26] and concatenate all POS tags together as a text feature.

Parsing Tree A parsing tree is a typical representation of the grammatical structure of a sentence [35]. It shows how a sentence is comprised. Each node represents a grammar unit, and its children are subunits that it is comprised of. Figure 2 illustrates a parsing tree for a sample review sentence "*The user interface is not very elegant*", which is generated by the Stanford Parser [3]. The label in each node denotes a POS tag. This tree means that the sentence (ROOT) is constituted by a noun phrase (NP) and a sub-sentence (S), where the noun phrase is constituted with a determiner (DT) and two nouns (NN).

In order to represent a parsing tree as a flat text feature, we traverse tree nodes in the breadth first order and select the first five nodes. We concatenate the POS tags of these five nodes as the text feature. For example, the feature for the parsing tree in Figure 2 is "ROOT-NP-S-DT-NN-NN".

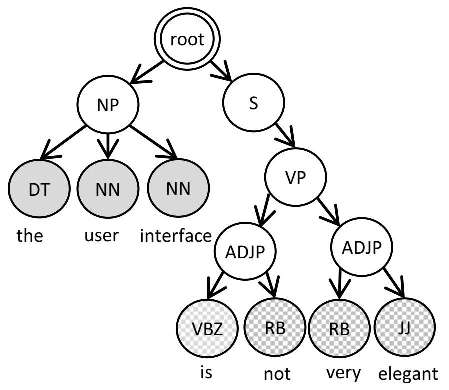


Fig. 2: Parsing tree for the sentence: *The user interface is not very elegant.*

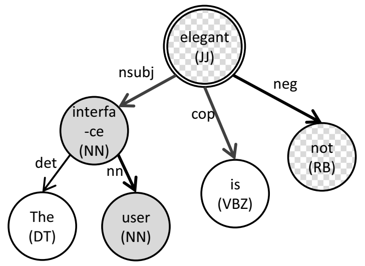


Fig. 3: Semantic Dependence Graph for the sentence: *The user interface is not elegant.*

Semantic Dependence Graph (SDG) Semantic dependence graph (SDG) [12] exhibits the semantic dependence among words in one sentence. It is a directed graph [12]. Nodes in the graph represent words and the corresponding POS tags. Edges represent semantic relations between words (e.g., noun subjection and adjective modifier). Each SDG has a root node which has no incoming edges. Figure 3 illustrates an SDG of a sample review sentence "*The user interface is not elegant*", which is generated by the Stanford Parser [3]. The root node is the word *elegant* which is an adjective (noted as JJ). It has three children: a noun subjection (nsubj) *interface*, a copula (cop) *is* and a negation modifier (neg) *not*. The child *interface* also has two children: a determiner (det) *the* and a noun compound modifier (nn) *user*.

To convert an SDG into a flat text feature, we traverse its nodes in breadth first order, then concatenate edges and POS tags in the traversal. We ignore leaves that are not linked to the root. For example, the feature for the SDG in Figure 3 is "VBZ-nsubj-NN-cop-VBZ-neg-RB".

C. Step 3 - Aspect-Opinion Extraction

Our next goal is to summarize users' opinions toward corresponding aspects. To do that, we need to identify words that express aspects and words that express opinions toward these aspects. In this step, SUR-Miner extracts aspect-opinion pairs (i.e., aspect and opinion words) from each review sentence classified in the *aspect evaluation* category. For example, the resulting aspect-opinion pairs for the review sentence "*The Prediction is accurate, but the auto-correct is annoying*" are: ⟨*prediction, accuracy*⟩ and ⟨*auto-correct, annoying*⟩.

In general, the state-of-the-art techniques extract aspects by frequent item mining or by topic model which views user reviews as bags of words [6], [10], [15]. Such an assumption may be problematic for software reviews that exhibit multiple purposes and sentiments.

As an empirical study indicates, software reviews have quite monotonous patterns for different purposes [31]. Therefore, it is possible to determine aspect-opinion pairs from the sentence patterns directly. Based on this assumption, we design a pattern-based parsing method which makes use of the syntax and semantics of review sentences and parses aspects and corresponding opinions from them directly. To do that, we first apply an NLP parser to annotate a semantic dependence graph (SDG) [12] for a review sentence. Then, we build a pattern-based parser to extract aspect-opinion pairs from the SDG.

1) Pattern-based Parsing: Our pattern-based parser is implemented as a sequence of cascading finite state machines [7]. The parser accepts a SDG and identifies aspect-opinion pairs based on predefined semantic templates.

Table II lists some typical semantic templates we use. The two letters at the beginning (e.g., JJ and NN) represent the POS tag of the root. Words in the following round brackets (e.g., *have* and *like*) represent root words. The children of the root are listed in the square brackets as edge-POS pairs. For example, the template in the first row means a root node with a POS tag of JJ and two children: a noun subjection (nsubj) with a POS tag of NN and a copula (cop) with a POS tag of VBZ. We generated the templates by manually identifying aspect part and opinion part from review sentences. We randomly selected 2,000 reviews sentences labeled as *Aspect Evaluation* except those we later used for evaluating the accuracy. First, we went through all these sentences and generated their SDGs. Then, we associated each SDG with a template which denotes the places of the aspect part and the opinion part in the SDG. We selected all those templates which were associated with more than 10 sentences in order to avoid accidental associations. We identified 26 such templates to design the finite state machine[3].

Then, given a new SDG instance, the parser travels from the root to all other nodes, checking the nodes, edges, and the corresponding children to determine the aspect and opinion words according to the templates. For example, given the SDG in Figure 3, the parser checks the POS tag of the root. Since it is an adjective (JJ) that matches the first and second templates in Table II, it further checks whether it has three children: a noun subjection (nsubj) with a POS tag of noun (NN), a copula (cop) with a POS tag of VBZ, and a negation modifier (neg) with a POS tag of RB. The second template is matched. Then, it checks whether the first child has a child of noun compound modifier (nn) with a POS tag of noun(NN). As the second template is an absolute match with the sample SDG, the parser recognizes the nsubj-NN node *interface* with its child *user* as aspect words and the neg-RB node *not* together with the root node *elegant* as opinion words.

D. Step 4 - Aspect Sentiment Analysis

In addition to opinions, a quantitative summarization of users' feeling towards each aspect may also be useful to grasp users' preferences. Users' ratings can provide such summarization objectively. However, an overall rating cannot satisfactorily characterize users' preferences for different aspects. For example, consider the review "*The UI is nice but the sound sucks.*" with a rating of 2 (out of 5). The user obviously likes the aspect *UI* but dislikes the aspect *sound*. Therefore, the actual ratings for both two aspects cannot be 2; it could be 3 for *UI* and 1 for *sound*.

At the fourth step, we apply sentiment analysis for each review sentence and associate the sentiments to the corresponding aspects with user ratings and a sentiment analysis tool. We first apply a state-of-the-art sentiment analysis tool *Deeply Moving* [1] to analyze sentiment for each review sentence. The *Deeply Moving* produces sentiments in a 0 to 4 scale, where 4 represents strongly positive, 0 means strongly negative and 2 means neutral. Then, to improve accuracy, we adjust the sentiments by user rating (1 to 5). Specifically, if the rating for a whole review is 5 (strongly positive), we add 1 to the sentiments of 0. If the rating is 1 (strongly negative), we minus 1 to the sentiments of 4.

For example, the following review has two sentences: *The interface is beautiful. I don't like the theme.* The sentiments for the two sentences are 4 and 0, respectively. If the user rating for the review is 5, we adjust the sentiment for the second sentence to 1 (= 0+1). If the user rating is 1, we adjust the sentiment for the first sentence to 3 (= 4-1).

E. Step 5 - Aspect Clustering and Summarization

At this step, we group aspect-opinion pairs with the same aspects and summarize sentiments and typical opinions for each aspect group.

To group the aspects, we first mine frequent items for all aspect words, namely, aspect words in the extracted aspect-opinion pairs. Then, we cluster aspect-opinion pairs with common frequent items (words). For example, given that *auto correct* is a frequent item in all aspect words, if there are two aspect-opinion pairs which contain this item, they will be clustered into one group. In particular, if a pair has two or more frequent items that can be clustered into more than two different groups, we cluster it into the group with the highest frequency of items or words. For example, a pair ⟨*background color, nice*⟩ can be grouped with both

[3]The full list of templates is at http://www.cse.ust.hk/~xguaa/srminer/appendix.html

TABLE II: Examples of dependency relation templates

Templates	Sample Sentence	Aspect Words	Opinion Words
JJ[nsubj-NN,cop-VBZ]	The UI is beautiful!	nsubj-NN	JJ
JJ[nsubj-NN[nn-NN],cop-VBZ,neg-RB]	The user interface is not elegant.	nn-NN + nsubj-NN	neg-RB + JJ
NN[amod-JJ]	nice UI!	NN	amod-JJ
VB(have)[nsubj-NN,nobj-NN]	The frame has nice UI!	nsubj-NN	have + nobj-NN
VB(like)[nsubj(I),nobj-NN]	I like the UI!	nobj-NN	like

⟨*background, beautiful*⟩ and ⟨*color, disgusting*⟩. However, if we have already known that the aspect *background* has a higher frequency than that of *color*, we will group the first pair with the second one instead of the third one. If there is no frequent item in the two aspect-opinion pairs, we group them together when they have common words in their aspects.

For each group, we select a *group keyword* as the word or the item which has the largest frequency in that group. We also calculate a *group sentiment* as the average adjusted sentiment of aspect-opinion pairs in that group.

F. Step 6 - Visualization

We designed two interactive diagrams, namely, *Aspect Heat Map* and *Aspect Trend Map*, to illustrate the summaries.

The *Aspect Heat Map* demonstrates popular aspects that users are concerned with. It aims to help developers and managers grasp which parts (aspects) of the app are loved or disliked by users. Figure 4 shows an example of the Aspect Heat Map with each circle indicating an aspect. The larger the circle is, the more popular and liked the aspect is. We define the size of the circle as $size = log(\#comments) + sentiment$. The horizontal axis represents the number of comments, and the vertical axis represents the adjusted rating. Therefore, circles in the top right represent most popular and loved aspects, and vice versa. To get insight into an aspect group, developers can click each aspect (circle) to view the specific comments with the top positive and top negative sentiments. For each comment, the aspect words are underlined and the opinion words are in bold.

The *Aspect Trend Map* demonstrates the sentiment trends over time. Capturing user reactions is important for developers to select and prioritize features [16], [34]. The *Aspect Trend Map* aims to help developers assess whether their recent changes affected users' satisfaction. It also enables developers to estimate and predict users' preferences so that they can improve parts of their product in the future. Figure 5 shows an example of the diagram with each line indicating the sentiment trend for a popular aspect. The horizontal axis represents date, and the vertical axis represents user sentiments.

Both the Aspect Heat Map and Aspect Trend Map are available on our project website at **http://www.cse.ust.hk/~xguaa/srminer/**.

IV. EMPIRICAL EVALUATION

We evaluate our framework through three dimensions: effectiveness, comparison and usefulness. To evaluate the effectiveness and advantages, we apply common measures in the text mining literature and compare the results with state-of-the-art methods. We also conduct developer surveys to evaluate the usefulness. Specifically, our evaluation addresses the following research questions:

- **RQ1** (effectiveness): How effectively can SUR-Miner classify reviews, extract aspects and opinions, and analyze sentiments for app reviews?
- **RQ2** (comparison): How does the SUR-Miner compare to state-of-the-art techniques for app review summarization?
- **RQ3** (usefulness): How the summaries by SUR-Miner useful for developers?

TABLE III: Overview of App Subjects

Data Set	Category	Time Period		
Swiftkey	productivity	8.26.2014	–	9.10.2014
Camera360	photography	8.24.2014	–	9.8.2014
Templerun2	game	8.30.2014	–	9.10.2014
WeChat	social network	9.5.2014	–	9.11.2014
KakaoTalk	communication	6.22.2014	–	9.12.2014
GooglePlayBooks	books	12.16.2014	–	3.18.2015
SpotifyMusic	music	3.8.2015	–	3.18.2015
YahooWeather	weather	1.30.2015	–	3.18.2015
GoogleMap	map	3.6.2015	–	3.20.2015
GoogleCalendar	productivity	2.4.2015	–	3.20.2015
ESPN	sports	9.19.2014	–	3.21.2015
TextPlus	social	12.16.2014	–	3.21.2015
Duolingo	education	3.5.2015	–	3.22.2015
Chasemobile	finance	9.17.2014	–	3.23.2015
Medscape	medical	1.7.2013	–	3.23.2015
Yelp	food	12.8.2014	–	3.25.2015
IMDB	entertainment	10.13.2014	–	3.29.2015

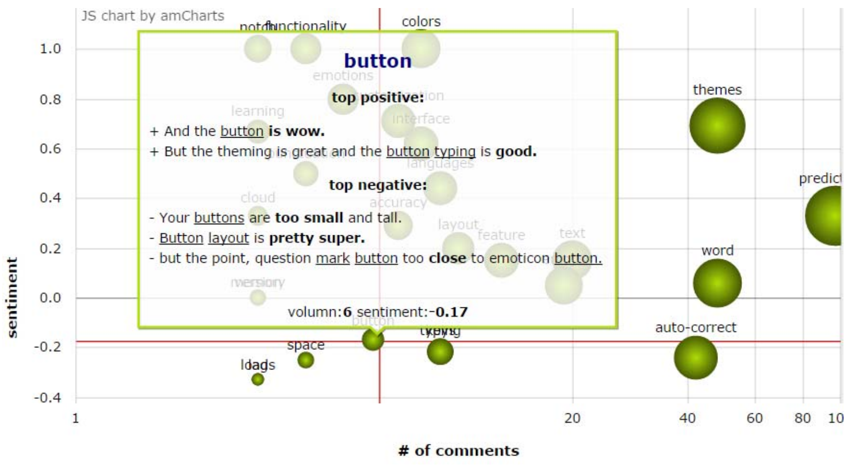
A. Data Collection

We choose 17 popular Android apps such as Swiftkey, Camera360, WeChat and Templerun2 from Google Play as our subjects. These apps cover 16 most popular categories such as games, communication, books and music. We collected the reviews roughly in the period from Aug, 2014 to Mar, 2015 using an open-source Android market API [2]. For each review, we collect its timestamp, rating, title and content. Table III shows the description of the subjects.

B. Effectiveness (RQ1)

In this section, we present our evaluation of SUR-Miner's effectiveness in each single step, namely, review classification, aspect-opinion extraction, and sentiment analysis.

1) Review Classification: First, we evaluate SUR-Miner on the review classification task. We sampled 2,000 review sentences from each dataset and compared the predicted results with golden standard labels. We manually labeled golden standard classes according to the rules in Table I. To reduce the labeling bias, two researchers separately applied the labeling rules to the 2,000 review sentences. Consensus labels



button

top positive:

+ And the button **is wow.**
+ But the theming is great and the button typing is **good.**

top negative:

- Your buttons are **too small** and tall.
- Button layout is **pretty super.**
- but the point, question mark button too **close** to emoticon button.

volumn:**6** sentiment:**-0.17**

Fig. 4: Demonstration of Aspect Heat Map

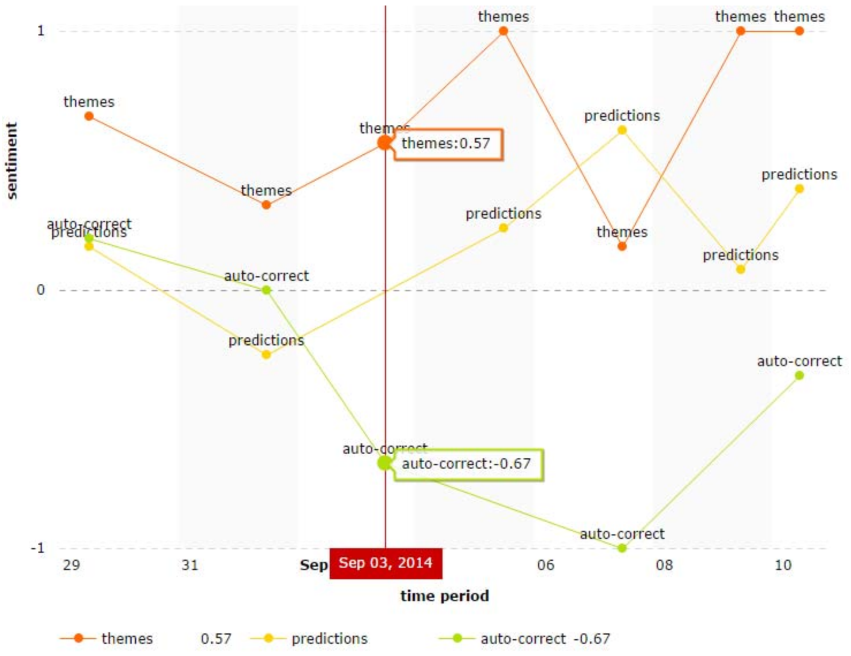


Fig. 5: Demonstration of Aspect Trend Map

were selected in the first iteration. For the disagreements, we discussed and clarified our labeling rules and relabeled again. A second iteration resulted in 100% agreement between the two researchers.

We use F1-score to measure the classification accuracy. The F1-score is widely used in the text classification literature [16], [38]. It is defined as follows

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{1}$$

where the precision is the ratio of the number of instances correctly classified as a class (TP) to the number of instances classified as the class (TP+FP).

$$precision = \frac{TP}{TP + FP} \tag{2}$$

The recall is the ratio of the number of instances correctly classified as a class (TP) to the number of instances in the class (TP+FN).

$$recall = \frac{TP}{TP + FN} \tag{3}$$

We performed a five-fold cross validation [38] in the data sets 100 times with each folder containing 400 review sentences.

Table IV shows the F1-scores for different categories[4]. Each column shows the F1-scores of a review category in all subjects. The last column averages the results for each subject in all review categories, and the last row averages F1-scores for each review category in all subjects. As indicated in the table, the classification performance is reasonable (with an

[4]Full results including precisions and recalls are at http://www.cse.ust.hk/~xguaa/srminer/appendix.html

TABLE IV: F1-scores of review classification in all subjects

Category	Evaluation	Praise	Request	Bug	Other	Overall
Swiftkey	0.72	0.87	0.78	0.58	0.86	0.76
Camera360	0.72	0.95	0.42	0.76	0.85	0.74
Templerun2	0.76	0.83	0.65	0.82	0.77	0.77
WeChat	0.70	0.93	0.50	0.76	0.89	0.76
KakaoTalk	0.76	0.96	0.66	0.47	0.91	0.75
GooglePlayBooks	0.59	0.92	0.72	0.60	0.85	0.74
SpotifyMusic	0.68	0.94	0.54	0.57	0.87	0.72
YahooWeather	0.83	0.94	0.57	0.56	0.87	0.76
GoogleMap	0.73	0.88	0.60	0.76	0.84	0.76
GoogleCalendar	0.74	0.77	0.80	0.70	0.82	0.77
ESPN	0.77	0.80	0.57	0.77	0.83	0.75
TextPlus	0.65	0.94	0.41	0.72	0.88	0.72
Duolingo	0.79	0.95	0.67	0.50	0.88	0.76
Chasemobile	0.75	0.93	0.46	0.56	0.85	0.71
Medscape	0.84	0.94	0.63	0.71	0.88	0.8
Yelp	0.77	0.91	0.40	0.58	0.91	0.72
IMDB	0.71	0.90	0.60	0.64	0.84	0.74
Average	0.74	0.90	0.59	0.65	0.86	0.75

TABLE V: F1-scores for aspect-opinion extraction

Data Set	Aspect	Opinion	Sentiment Positive	Sentiment Negative
Swiftkey	0.87	0.86	0.87	0.71
Camera360	0.87	0.87	0.89	0.53
Templerun2	0.95	0.93	0.91	0.79
WeChat	0.83	0.82	0.77	0.83
KakaoTalk	0.84	0.87	0.85	0.77
GooglePlayBooks	0.84	0.86	0.82	0.82
SpotifyMusic	0.84	0.86	0.83	0.62
YahooWeather	0.90	0.63	0.89	0.77
GoogleMap	0.86	0.84	0.88	0.88
GoogleCalendar	0.79	0.82	0.78	0.85
ESPN	0.80	0.78	0.69	0.83
TextPlus	0.84	0.85	0.80	0.77
Duolingo	0.86	0.85	0.93	0.51
Chasemobile	0.84	0.88	0.87	0.77
Medscape	0.89	0.90	0.86	0.60
Yelp	0.84	0.87	0.89	0.82
IMDB	0.84	0.87	0.86	0.86
Average	0.85	0.84	0.85	0.75

average F1-score of 0.75) as well as for the *aspect evaluation* category (with an average F1-score of 0.74). That means the classification step can accurately provide different developers with different types of review sentences. In particular, it provides reliable review sentences for the aspect evaluation. The F1-scores for specific categories such as "bug" are not good in some apps. We manually checked those reviews and found that these apps received rare bug reports. The extremely unbalanced data could be the main reason for these outliers.

2) Aspect-Opinion Extraction: To evaluate SUR-Miner's performance on aspect-opinion extraction, we follow the same procedures as in the review classification experiment to check if SUR-Miner correctly extracts aspects and corresponding opinions from review sentences. For each subject, we sampled 2,000 review sentences and selected those in the *Aspect Evaluation* category. We use F1-score to measure the accuracy of aspect extraction and opinion extraction separately. In particular, the number of true positives (TP) in Equation 1-3 is the number of correctly extracted aspects or opinions; the number of false positives (FP) means the number of mistakenly extracted aspects or opinions; the number of false negatives (FN) is defined as the number of aspects or opinions that have not been extracted.

The results are shown in the first two columns in Table V. As is indicated, both aspect extraction and opinion extraction have reasonable accuracy, with average F1-scores of 0.85 and 0.84, respectively[4]. The results suggest that the aspect extraction step provides reliable aspects and opinions.

3) Sentiment Analysis: To evaluate the sentiment analysis step, we also follow the same procedures as in the classification and aspect extraction stages. For each subject, we sampled 2,000 review sentences and selected those in the *Aspect Evaluation* category, and compared the sentiment for each aspect-opinion pair with golden standard sentiment labels. To simplify the estimation, we divided the sentiment scale (0-4) into two polarities, that is, positive (3-4) and negative (0-1) [32], and labeled them according the their polarities. We labeled the golden standard sentiments manually as we did for review classification.

We use F1-score to measure the accuracy of each sentiment category. In particular, the number of true positives (TP) in Equation 1-3 is defined as the number of correctly classified sentiments; the number of false positives (FP) means the number of misclassified sentiments; the number of false negative (FN) means the number of sentiments that are not classified in that category.

The results are shown in the last two columns in Table V. As is indicated, both positive and negative sentiments have acceptable accuracy, with average F1-scores of 0.85 and 0.75, respectively[4]. The average F1-score for both is 0.80. The reason that the negative sentiment has a relatively low performance in Camera360 and Duolingo could be that these two apps received much more positive reviews so that the sentiment categories become extremely unbalanced. The results suggest that the sentiment analysis step produces reliable results.

SUR-Miner provides reliable results on review classification, aspect-opinion extraction, and sentiment analysis, with average F1-scores of 0.75, 0.85 and 0.80, respectively.

C. Comparison (RQ2)

Our next evaluation aims to compare SUR-Miner with state-of-the-art techniques with respect to final summaries.

1) Quantitative Comparison: We first compare the accuracy of SUR-Miner for aspect extraction with those of related work: ReviewSpotlight [37] and Guzman's method [17]. As discussed in Section II, ReviewSpotlight is a review summarization tool for general products by identifying noun-adjective pairs (Section II-C), and Guzmans' tool is the most related work to ours that also extracts aspects from app user reviews (Section II-B).

We run aspect extraction by simulating real world usage scenarios. For each subject, we randomly select 400 review sentences in all categories from the original dataset except those for training classifiers. First, we run review classification on these sentences. Then, we apply aspect extraction on sentences that are classified as *Aspect Evaluation*. We compare the extracted aspects with golden standard aspects that were manually labeled. We use F1-score to evaluate the accuracy using the same definition in Section IV-B2.

TABLE VI: Comparison of Aspect Extraction Accuracy with Related Works

Metric	SUR-Miner	ReviewSpotlight	Guzman and Maalej [17]
F1-score	0.81	0.56	0.55

Table VI shows the average F1-scores of three approaches in all subjects. We reproduced ReviewSpotlight and applied it for extracting app aspects. The result of Guzmans' approach is excerpted from their paper [17]. As we can see, the F1-score for SUR-Miner is 0.81, significantly greater than those of the ReviewSpotlight (0.56) and Guzman's tool (0.55).

To investigate the reasons for these results, we manually checked the results of ReviewSpotlight. We found that without distinguishing review categories, it tends to extract aspects for reviews in other categories such as *aspect requests* and *bug reports*. For example, consider the review *"I hate that you can't use offline dictionary"* which requires for a new aspect *offline dictionary*. The ReviewSpotlight just outputs ⟨*dictionary, offline*⟩ which is meaningless while SUR-Miner can filter such review from *aspect evaluation* since it talks about a nonexistent aspect.

Another shortcoming of these related approaches is that, they cannot identify complex phrases as they simply consider frequent items or noun-adjective pairs as aspects. For example, for the review *"Also, love the way it auto ads reminders"* , the ReviewSpotlight simply outputs $\langle ads, auto \rangle$ while SUR-Miner outputs ⟨ *the way it auto ads reminers, love*⟩.

It is also interesting to see that even though both the classification and extraction stages have mistakes, combining them does not result in worse accuracy. The classification step has an F1-score of 0.74. The aspect extraction step has an F1-score of 0.85 (Section IV-B). However, when extracting aspects from the outputs of classification stage, the final F1-score is 0.81, even greater than that in the classification stage. By manually checking the extracted aspects, we found that though some reviews were misclassified during the classification stage, the aspect extraction stage can still "re-correct" them since a misclassified review may not be parsed by our semantic patterns. For example, consider a misclassified review "No public transportation navigation!" which requires a new aspect but was misclassified as *Aspect Evaluation* in the classification stage. Nevertheless, SUR-Miner still cannot recognize any aspect since there is no semantic pattern to parse this review.

2) Qualitative Comparison: Topic models such as LDA are widely used by most state-of-the-art app review summarization tools [10], [15], [17]. To investigate the advantages of SUR-Miner over these topic-based techniques, we qualitatively compare the extracted aspects by SUR-Miner with topics extracted by topic models.

Table VIII compares the top five aspects we extracted with the top five topics by AR-Miner (a state-of-the-art review summarization tool that applies EMNB-LDA topic model) [10] in the *Swiftkey* subject. We collected data in the same period from Google Play as AR-Miner did. We have two observations: 1) SUR-Miner can distinguish different review purposes. For example, opinions extracted by SUR-Miner are aspect evaluations except some noises, while top words by LDA (AR-Miner) are miscellaneous. For example, if a manager would like to know users' evaluations on the aspect *prediction*, SUR-Miner can provide users' opinions such as *excellent*, *accurate*, *hate* while AR-Miner (LDA) cannot provide such information; 2) SUR-Miner can distinguish users' sentiments while AR-Miner (LDA) cannot. For example, managers and developers can find both positive and negative sentiments by SUR-Miner but cannot tell whether user like or dislike the aspect *prediction* by LDA.

TABLE VIII: Comparison of topics (by LDA) and aspects (by SUR-Miner)

(a) Topics and typical words by AR-Miner (LDA) [10]. The first row lists the top 5 topics. The following 4 rows list the top 5 words for each topic

Topics	theme	Chinese	jelly bean	predict	space
Keywords	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

(b) Aspects and opinions extracted by SUR-Miner. The first row lists top five aspects with most comments. The following three rows show opinions with top positive sentiments while the last two rows show opinions with top negative sentiments

Aspects	predictions.	auto-correct.	words	theme	key
Opinions	amazing	flawless	love	great	best
	excellent	good	like	love	like
	amazing	amazing	like	over top	
	accurate	stubborn	a pain	ugly	breaker
	hate	nightmare	not-need	just	obnoxious

Overall, SUR-Miner produces much clearer summaries in distinguishing review purposes and sentiments than LDA models.

SUR-Miner produces much more accurate and clearer summaries than state-of-the-art methods.

D. Usefulness (RQ3)

As the usefulness evaluation may be subjective, we consulted developers to assess the usefulness of SUR-Miner. We applied SUR-Miner to the latest user reviews of 17 popular Android apps such as Swiftkey, Camera360, WeChat and Templerun2. We presented the visualized summaries as demos in our website and asked the questions shown in Table VII to developers. The two questions are related to the two diagrams respectively. We provided five options for each of them (5 strongly agree, 4 agree, 3 neither, 2 disagree and 1 strongly disagree). The number of choices to each option was also listed for each question.

We sent invitation mails to developers of the selected apps, posted our website to Android developer communities in Google+, and also invited developers from IT companies such as Samsung, Tencent and Baidu for feedback.

Developers showed great interest in our SUR-Miner. As indicated in Table VII, of all 32 answers received, 28 of them (88%) agreed that our tool helps developers. Only two held conservative opinions (6.3%) and two (6.3%) disagreed. Figure 6 shows the box plot statistics of developer feedback.

TABLE VII: Questions in the developer survey and results

Questions	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree	Total
Q1. Do you think Figure 1 "Aspect Heat Map" is useful to understand users preferences for aspects?	0	1	0	7	8	22
Q2. Do you think Figure 2. "Aspect Trend Map" helps developers to understand the users preferences trend over time?	0	1	2	5	8	22

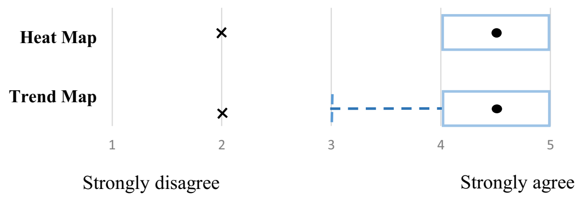


Fig. 6: Developer Rating on Usefulness

We quantize the answers to ratings from 1 to 5. Each box shows answers for a question. As the results indicate, answers to both questions have average ratings much greater than 3. That means developers agree the usefulness of SUR-Miner in general.

In addition, we received the following encouraging comments from developers:

"This is a great project. The visualization data impresses me much!"

"I think if possible, we would like to work with these researchers. I really like the performance of your sentiment classifier."

"The provided visualized information serves as quite clear way to get insight of products including advantages and disadvantages. Analyzing large scale user comments calls for great human efforts. Such project makes the understanding as well as iteration of products fast."

These comments indicate that developers appreciate our tool to help grasp users' opinions toward different aspects.

Developers feedback indicates our SUR-Miner helps developers grasp users' opinions and sentiments in practice.

V. THREATS TO VALIDITY

We have identified the following threats to validity:
Subjects are all free Android apps. All projects investigated in this paper are free Android apps. Hence, they might not be representative of charged apps and apps in other markets (e.g., AppStore) [27]. Commercial apps may have different review patterns. In the future, we will mitigate this threat by investigating user reviews of commercial apps and apps in other markets.
Ground truth labels were judged by two people. As the golden standard labels require large human efforts, they were judged by only two people in our experiments. They could be biased from real app developers. For mitigating this threat we presented final results to developers and made sure that they were satisfied with the accuracy. In the future, we will further reduce this threat by inviting more developers for labeling.

VI. CONCLUSION

We proposed SUR-Miner for effective and automatic user review summarization. The summaries from SUR-Miner provide a desirable answer to the important question "*What part of your apps are loved by users*" for developers.

Our evaluation results show that SUR-Miner provides reliable results, with average F1-scores of 0.75, 0.85 and 0.80 for review classification, aspect-opinion extraction and sentiment analysis, respectively. The final aspects from SUR-Miner are significantly more accurate and clearer than state-of-the-art techniques, with an F1-score of 0.81, greater than that of ReviewSpotlight (0.56) and Guzmans' method (0.55). Feedback from app developers is also very encouraging, and 88% answers from developers agree with the usefulness of SUR-Miner.

In the future, we will summarize other review categories such as *feature requests*. In addition, we will propose techniques to summarize other software text data such as code comments and bug reports.

REFERENCES

[1] Deeply moving: Deep learning for sentiment analysis. http://www-nlp.stanford.edu/sentiment/. retrieved june 02,2014.

[2] An open-source api for the android market. https://code.google.com/p/android-market-api/. retrieved september 01,2014.

[3] The stanford parser: A statistical parser. http://nlp.stanford.edu/software/lex-parser.shtml. retrieved june 02,2014.

[4] T. Abou-Assaleh, N. Cercone, V. Kešelj, and R. Sweidan. N-gram-based detection of new malicious code. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 41–42, 2004.

[5] A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, pages 12–23, 2014.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[7] B. K. Boguraev. Towards finite-state analysis of lexical cohesion. In *Proceedings of the 3rd international conference on finite-state methods for NLP*, 2000.

[8] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[9] W. B. Cavnar, J. M. Trenkle, et al. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[10] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, pages 767–778, 2014.

[11] D. Das and S. Petrov. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 600–609, 2011.

[12] M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454, 2006.

[13] A. Di Sorbo, S. Panichella, C. Visaggio, M. Di Penta, G. Canfora, and H. Gall. Development emails content analyzer: Intention mining in developer discussions. In *30th international conference on Automated Software Engineering (ASE 2015). Lincoln, Nebraska*, 2015.

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

[15] 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.

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

[17] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 153–162, 2014.

[18] J. Houvardas and E. Stamatatos. N-gram feature selection for authorship identification. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 77–86. Springer, 2006.

[19] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177, 2004.

[20] J. Huang, O. Etzioni, L. Zettlemoyer, K. Clark, and C. Lee. Revminer: An extractive interface for navigating reviews on a smartphone. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 3–12, 2012.

[21] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 41–44, 2013.

[22] Y. Jo and A. H. Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 815–824, 2011.

[23] V. Kešelj, F. Peng, N. Cercone, and C. Thomas. N-gram-based author profiles for authorship attribution. In *Proceedings of the conference pacific association for computational linguistics, PACLING*, volume 3, pages 255–264, 2003.

[24] R. Konig, R. Renner, and C. Schaffner. The operational meaning of min-and max-entropy. *Information Theory, IEEE Transactions on*, 55(9):4337–4347, 2009.

[25] S. Lahiri and R. Mihalcea. Using n-gram and word network features for native language identification. In *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 251–259, 2013.

[26] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[27] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *Mining Software Repositories (MSR), 12th IEEE Working Conference on*, 2015.

[28] A. McCallum, D. Freitag, and F. C. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598, 2000.

[29] J. Nichols, M. Zhou, H. Yang, J.-H. Kang, and X. H. Sun. Analyzing the quality of information solicited from targeted strangers on social media. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 967–976, 2013.

[30] D. Pagano and B. Bruegge. User involvement in software evolution practice: a case study. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 953–962, 2013.

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

[32] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[33] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of the 31st International Conference on Software Maintenance and Evolution (ICSME 2015). Bremen, Germany*.

[34] 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 *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, 2015.

[35] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*, 2013.

[36] A. Sureka and P. Jalote. Detecting duplicate bug report using character n-gram-based features. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 366–374, 2010.

[37] K. Yatani, M. Novati, A. Trusty, and K. N. Truong. Review spotlight: a user interface for summarizing user-generated reviews using adjective-noun word pairs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1541–1550, 2011.

[38] 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.