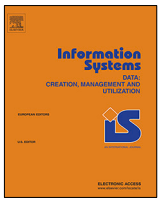
Information Systems 114 (2023) 102181

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Mining and searching app reviews for requirements engineering: Evaluation and replication studies

Jacek Dąbrowski [a,b,*], Emmanuel Letier [a], Anna Perini [b], Angelo Susi [b]

[a] University College London, London, UK
[b] Fondazione Bruno Kessler, Trento, Italy

ARTICLE INFO

Article history:
Received 23 April 2021
Received in revised form 2 January 2023
Accepted 22 January 2023
Available online 31 January 2023
Recommended by Gottfried Vossen

Dataset link: https://github.com/jsdabrowski/IS-22/

Keywords:
Mining user reviews
Software engineering
Feature extraction
Sentiment analysis
Searching for feature-related reviews
Empirical study

ABSTRACT

App reviews provide a rich source of feature-related information that can support requirement engineering activities. Analyzing them manually to find this information, however, is challenging due to their large quantity and noisy nature. To overcome the problem, automated approaches have been proposed for 'feature-specific analysis'. Unfortunately, the effectiveness of these approaches has been evaluated using different methods and datasets. Replicating these studies to confirm their results and to provide benchmarks of different approaches is a challenging problem. We address the problem by extending previous evaluations and performing a comparison of these approaches. In this paper, we present two empirical studies. In the first study, we evaluate opinion mining approaches; the approaches extract features discussed in app reviews and identify their associated sentiments. In the second study, we evaluate approaches searching for feature-related reviews. The approaches search for users' feedback pertinent to a particular feature. The results of both studies show these approaches achieve lower effectiveness than reported originally, and raise an important question about their practical use.

1. Introduction

App stores have become important distribution platforms of mobile apps [1]. In 2022, Apple Store and Google Play Store facilitate more than 5 million apps; and are commonly used to discover, download and update software products [2]. These App Stores have made a substantial impact on software engineering practices, in particular by bridging the gap between users and app developers [3].

App reviews are user-written comments enriched with a star rating that app users can facilitate to other App Store users and app developers about their experience of an app [3–5]. The majority of app reviews have length up to 675 characters [6]; and are rich source of feature-related information such as user opinions about app features, description of usage scenarios as well as different types of user requests [1,3–5,7].

This feedback can help developers to understand how users perceive their app, as well as to identify the users' requirements and preferences [3,4,8,9]. Surveys of software developers have shown that identifying what users say about specific app features is an important concern for developers [5,7,9,10]. This information affects multiple software engineering activities, from requirements engineering to testing and system maintenance and evolution [3,4,7,8].

Analyzing app reviews to find feature-related information, however, is challenging due to their large number and the difficulty in extracting actionable information from short informal texts [3,11]. Popular apps like WhatsApp Messenger can receive more than 5,000 reviews per day [6,12]; moreover, the review content can vary from informative and helpful one to content conveying hate and spam [1,6]. Consequently, the possibility of using this information to support engineering activities is obstructed [3,8].

To address the problem, techniques have been proposed to searching for feature-related reviews [13–15], and mining user opinions about these features [15–19]. These approaches facilitate the analysis of a large amount of online user feedback by performing one of the following tasks, or a combination of them: searching for app reviews pertinent to a particular feature, extracting features discussed in reviews and identifying their associated users' sentiments [16,17].

In particular, two approaches have become adopted as a reference in the RE community, i.a., GuMa[1] [17] and SAFE [20], as they have been shown to achieve promising accuracy for performing

* Corresponding author at: University College London, London, UK.
E-mail address: jacek.dabrowski.16@alumni.ucl.ac.uk (J. Dąbrowski).

[1] We refer to the approach using abbreviations derived from their authors' surnames.

https://doi.org/10.1016/j.is.2023.102181

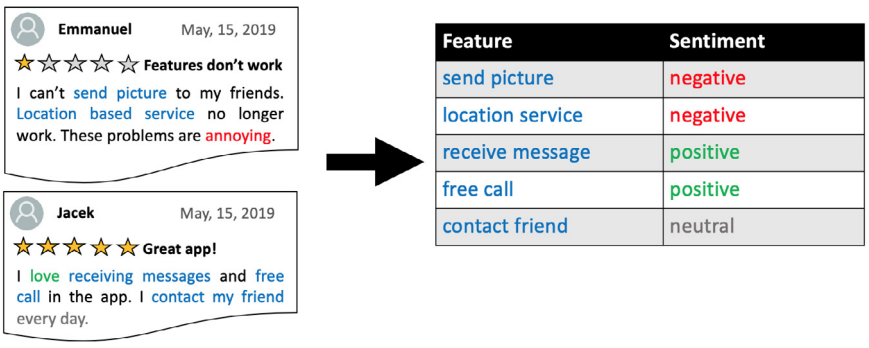


Fig. 1. Opinion Mining Task.

the task of opinion mining and searching for feature-related reviews [1,7].

The empirical evaluations of these approaches are however hard to replicate and to compare because they rely on different datasets and experiment designs. In some cases, the unavailability of their annotated datasets and the lack of details about their evaluation procedures are further challenges to the replicability of the experiments [15,17,20–22].

The aim of this paper is to address the problem by extending previous evaluations and performing a comparison of these app review analysis approaches. We consider the following research questions:

RQ1: What is the effectiveness of feature extraction approaches?

RQ2: What is the effectiveness of feature-specific sentiment analysis approaches?

RQ3: What is the effectiveness of approaches in searching for app reviews pertinent to a particular feature?

To answer the first two questions, we conducted an empirical study in which we evaluated three opinion mining approaches: GuMa [17], SAFE [20] and ReUS [18]. We evaluated them in performing two tasks: feature extraction and sentiment analysis using a new annotated dataset we have constructed for this experiment. This work was first presented at the 32nd International Conference on Advanced Information Systems Engineering (CAiSE'20) [19]

This journal paper extends the conference paper with a new study that addresses the third research question. In this study, we evaluate three candidate approaches searching for feature-related reviews, namely Lucene [23], MARAM [14] and SAFE [15].

The main contributions presented in this article are: (i) two empirical studies expanding previous evaluations of opinion mining approaches and approaches searching for feature-related reviews; (ii) a comparison of these approaches in performing the following three tasks: feature extraction, feature-specific sentiment analysis and searching for app reviews pertinent to a particular feature, and (iii) a replication package [24] containing two new datasets: the first one consisting of 1,000 reviews annotated with 1,521 opinions, and the second one including 1,113 reviews annotated with 24 app features; as well as our re-implementation of GuMa, MARAM and Lucene-based tools.

The paper is structured as follows: Section 2 introduces terminology, defines the problems of opinion mining and searching for feature-related reviews, presents the approaches we evaluate to address the problems. Section 3 presents scenarios motivating these approaches. Section 4 presents the empirical study addressing *RQ*1 and *RQ*2, and Section 5 the study addressing *RQ*3. Section 6 discusses related works and Section 7 concludes the paper.

2. Background

This section introduces terminology and the formulation of the opinion mining and searching for feature-related reviews problems. It also outlines the evaluated approaches. We refer to this information throughout the paper.

2.1. Terminology and problem formulation

We define a *feature* as a user-visible functional attribute of an app: a functionality (e.g., send message), a module providing functional capabilities (e.g., user account) or a design component that can be utilized to perform tasks (e.g., UI). The software engineering literature is generally inconsistent about the feature definition; a part of the literature pertains to features as functional attributes (e.g., [25,26]), while the other defines features as both functional and non-functional attributes (e.g., [27]).

In this work, the feature definition is focused on functional attributes as the evaluated tools (see Section 2.2) are neither intended to analyze non-functional attributes (e.g., [15]), nor the studies proposing the tools provide sufficient evidence about their suitability for this purpose [17]; in fact, the surveyed literature [7] suggests that custom-built techniques need to be adopted for this purpose (e.g., [28,29]).

App reviews can describe features seen at a different level of abstraction, at a high-level (e.g., communicate with my friends) and at a low-level one (e.g., click send message button) [17]. A *feature expression* is a non-empty set of words $f = \{w_1, \ldots, w_m\}$ describing the actual feature in an app review; this definition uses a set of words rather than a multi-set for the feature description as neither our manual analysis of app reviews nor the literature suggests features are described using repeated words. Further on in the text, we will refer to a feature expression as a feature for the sake of simplicity.

Like other types of on-line user feedback [30], app reviews can convey information about user attitude towards features. We here define a *sentiment* s as a user attitude which can be either *positive*, *negative* or *neutral*; and an *opinion* as a tuple $o = (f, s)$, where f is a feature in a review r, s is a sentiment referencing to f in r.

The first empirical study (see Section 4) focuses on the *opinion mining* problem, where given a set of reviews $R = \{r\}$ on an app a, the problem is to find a multi-set of all the opinions $O = \{o\}$ in a set of reviews R; this definition refers to a multi-set of all the opinions as the same opinion can be given in many reviews. Fig. 1 illustrates the opinion mining problem. This problem can be decomposed into two sub-problems, feature extraction and feature-specific sentiment analysis.

The *feature extraction* problem is to find a multi-set of all the features $F = \{f\}$ in a set of reviews $R = \{r\}$ on an app a; whereas, in the *feature-specific sentiment analysis*, given a set of

Table 1
App review mining tools that we have evaluated in our two experiments.

	Criterion	GuMa [17]	SAFE [15]	ReUS [18]	MARAM [14]	Lucene [23]
Functionality	Feature Extraction (RQ1)	✓	✓	✓	–	–
	Sentiment Analysis (RQ2)	✓	–	✓	–	–
	Feature-Related Search (RQ3)	–	✓	–	✓	✓
Meta-data	Availability	–	✓	✓	–	✓
	Release Year	2014	2017	2019	2016	2010
	Technology	Python	Python	Java	Python	Java

pairs $\{(f, r)\}$ where f is a feature in a review r, the problem is to find a multi-set $S = \{s\}$ where s is a sentiment referring to f in r; these definitions refer to a multi-set of features and a multi-set of feature-specific sentiments as the same feature or the same feature-specific sentiment can be given in many reviews.

The second empirical study (see Section 5) focuses on the problem of *searching for feature-related reviews*, where given a set of reviews $R = \{r\}$ on an app a and a feature, the problem is to find the subset of reviews in R that refer to this feature (i.e., feature-related reviews).

2.2. Approaches for mining feature-related information

We have evaluated five app review mining tools in our two studies. Table 1 shows a comparison of the tools that we have considered in the studies. The tools are compared by different criteria, grouped into two overall categories: Functionality and Meta-data information. The first category includes criteria about the presence of a functionality in a tool that we aim to evaluate: feature extraction (RQ1), sentiment analysis (RQ2) and feature-related search (RQ3). The latter category includes meta-data information about the tools such as their availability, release year and technology in which the tools have been created. A "✓" in the table indicates that a tool satisfies a certain criteria, whereas "-" denotes a tool which does not satisfy it.

In the first study (for RQ1 and RQ2), we selected three approaches: GuMa [17], SAFE [15] and ReUS [18]. We selected GuMa and SAFE as they are state-of-the-art approaches widely known in RE community [1,7,21,31]. We also selected ReUS [18] as the approach achieves a competitive performance in the context of opinion mining and sentiment analysis research [16,18]. We also have its original implementation. In the second study (for RQ3), we chose: SAFE [15], MARAM [14] and Lucene [23]. We selected SAFE and MARAM as they are the only proposed approaches searching for feature-related reviews in the app store community [1,7] (except for our previous work [13]). We also include Lucene which is a well-known general purpose search engine that is commonly used as a baseline in the empirical evaluation of SE research [7].

We here provide detailed descriptions of the five tools that we have evaluated in our two studies:

1. **GuMa** performs feature extraction and feature-specific sentiment analysis. The tasks are performed independently of each other. To extract features, the approach relies on a collocation finding algorithm; the algorithm identifies expressions of multiple words which commonly co-occur in a set of documents [32]. For predicting sentiment, the approach uses the SentiStrength tool [33]. First, the approach predicts the sentiment of a sentence, then assigns sentiments to features in the sentence. Unfortunately, GuMa's source code and evaluation data set are not available. We have therefore re-implemented GuMa's tool in Python referring to their original description [24]. Like in the original study [17], we implemented feature extraction using the collocation finding algorithm provided in the NLTK toolkit [34]; and used the SentiStrength library to implement sentiment analysis. We tested our implementation is a consistent approximation of the GuMa's original implementation on examples from the original paper and produces the same outputs.
2. **SAFE** supports feature extraction, feature-specific review searching, but not sentiment analysis. The approach extracts features based on linguistics patterns, including 18 part-of-speech patterns and 5 sentence patterns. These patterns have been identified through manual analysis of app descriptions. The approach conducts two main steps to extract features from a review: text preprocessing and the application of the patterns. Text preprocessing includes tokenizing a review into sentences, filtering-out noisy sentences, and removing unnecessary words. The final step concerns the application of linguistic patterns to each sentence to extract app features. As for the searching task, SAFE compares queried features with those extracted from reviews using semantic similarity. To improve the performance, the tool uses query expansion using WordNet lexical database. We used the original Python-written implementation of the tool in our study.
3. **ReUS** exploits linguistics rules comprised of part-of-speech patterns and semantic dependency relations. These rules are used to parse a sentence and perform feature extraction and feature-specific sentiment analysis. Both tasks are performed at the same time. Given a sentence, the approach extracts a feature and an opinion word conveying a feature-specific sentiment. To determine the sentiment, the approach exploits lexical dictionaries. We used the original implementation of the Java-written tool, and set up it to identify one out of three sentiment polarities.
4. **MARAM** supports the task of searching for feature-related reviews. To this end, the approach exploits a simplistic model by representing both query and reviews as the bags (sets) of their words. It then computes the similarity score between a given query and reviews using Jaccard Similarity coefficient. The tool next outputs ranked reviews based on their similarity score with the query. Neither MARAM tool nor their source code are available. We have therefore re-implemented the tool rigorously following their description in the original study [14]. We re-implemented the tool from scratch in Python using NLTK library [24]. We tested that our implementation is consistent with MARAM's original implementation; and produces the same outputs as the original tools using examples in the original paper.
5. **Lucene** is a free and open-source search engine software library suitable for any application requiring full-text indexing and searching capability [23]. It is widely known for its usefulness in the implementation of internet search engines and local, single-site searching. Lucene combines Vector Space Model and the Boolean Model to determine how relevant a given document is to a user's query. We implemented a basic app review searching tool in Java that uses Lucene with its default setting [24]. We followed the original Lucene documentation to implement the searching capability [35].

3. Motivating scenarios

We describe three use cases in which the use of approaches for opinion mining and searching for feature-related reviews provide benefits. They are inspired by real-world scenarios, which previous research analyzed [3,4,7,8].

Use Case 1 (Validation by Users) Knowing what features users love or dislike can give product managers an idea about user acceptance of these features [3,8,9]; it can help them draw a conclusion whether invested effort was worth it [8]. Imagine the development team changed core features in WhatsApp (e.g. video call). The team may want to know what users say about these features so that they can fix any glitches and refine these features. Mining user opinions could help them answer What are the most problematic features? or How many users do report negative opinions about a concrete feature?

Use Case 2 (Supporting Requirements Elicitation) Suppose WhatsApp receives negative reviews about a feature (e.g. group chat). Reading thousands of reviews to analyze the problem can be infeasible. Using an opinion mining tool, developers could discover the issue within minutes. The tool could group reviews based on discussed features and associated user's sentiment. Developers could then examine reviews talking negatively about a specific feature (e.g. group chat). This could help them understand user concerns about a problematic feature, and potentially elicit new requirements. Similarly, the development team might have been tasked to define requirements for one of feature requests. Finding reviews referring to the feature will allow them to quickly identify What users have been saying about the feature. This cheap elicitation technique might be insufficient in itself or it might be a starting point for additional more expensive elicitation methods.

Use Case 3 (Supporting Requirements Prioritization) When added with statistics, user opinions can help developers prioritize their work [3,8,9]. Suppose the team is aware about problems with certain features which are commented negatively. Finding negative opinions mentioning these features could help them to compare how often these opinions appear, for how long these opinions have been made, and whether their frequency is increasing or decreasing. This information could provide evidence of their relative importance from a user perspective. Such information is not sufficient in itself to prioritize change request because the perspective of other stakeholders must be also taken into account, but it can provide useful evidence to partly inform such decisions.

For these scenarios having a tool that facilitates (i) opinion mining, (ii) searching for feature-related reviews and (iii) provides their summary with simple statistics could help the team to evolve their app.

4. Study I: Opinion mining

The first empirical study evaluated and compared the approaches addressing the problem of opinion mining (see Section 2.1). We here describe the design of the study, report and discuss the results, together with threats to validity of this study.

4.1. Empirical study design

This section describes the empirical study design we used to evaluate the selected approaches. We provide the research questions we aimed to answer, the manually annotated dataset and evaluation metrics used to this end.

Table 2
The overview of the subject apps.

App Name	Category	Platform	#Reviews
Evernote	Productivity	Amazon	4,832
Facebook	Social	Amazon	8,293
eBay	Shopping	Amazon	1,962
Netflix	Movies & TV	Amazon	14,310
Spotify Music	Audio & Music	Google Play	14,487
Photo Editor Pro	Photography	Google Play	7,690
Twitter	News & Magazines	Google Play	63,628
Whatsapp	Communication	Google Play	248,641

4.1.1. Research questions

The objective of the study was to evaluate and compare approaches mining opinions from app reviews. To this end, we formulated two research questions:

RQ1: What is the effectiveness of feature extraction approaches?

RQ2: What is the effectiveness of feature-specific sentiment analysis approaches?

In RQ1, we evaluated the capability of the approaches in correctly extracting features from app reviews. In RQ2, we investigated the degree to which the approaches can correctly predict sentiments associated with specific features. A conclusive method of measuring the correctness of extracted features/predicted sentiments is by relying on human judgment. We used our dataset in which opinions (feature-sentiment pairs) have been annotated by human-coders (see Section 4.1.2). We compared extracted features/predicted sentiments to those annotated in ground truth using automatic matching methods (see Section 4.1.3). In answering the questions, we report precision and recall.

4.1.2. Manually annotated dataset

This section describes the manually annotated dataset we created to answer RQ1 and RQ2 [24]. To create this datatset, we collected reviews from previously published datasets [36,37] and asked human-coders to annotate selected samples of these reviews.

(A) Data Collection

We have selected all the reviews from datasets used in previous review mining studies [36,37]. We selected these datasets because they include millions of English reviews from two popular app stores (i.e., Google Play and Amazon) for different apps, categories and period of times. We selected 8 apps from these datasets, 4 apps from Google Play and 4 from Amazon app stores. For each subject app, we also collected their description from the app store. Table 2 illustrates the summary of apps and their reviews we used in our study. We selected subject apps from different categories to make our results more generalizable. We believe that the selection of popular apps could help annotators to understand their features, and to reduce their effort during the annotation.

(B) Annotation Procedure

The objective of the procedure was to produce an annotated dataset that we use as ground truth to evaluate the quality of solutions produced by feature extraction and sentiment analysis approaches [38]. Fig. 2 illustrates the overview of the procedure. Given a sample of reviews, the task of human-coders was to label each review with features and their associated sentiments.

We started by elaborating a guideline describing the annotation procedure, the definition of concepts and examples. We then asked two human-coders[2] to label a random sample of reviews

[2] The first author and an external coder who has no relationship with this research. Both coders have an engineering background and programming experience

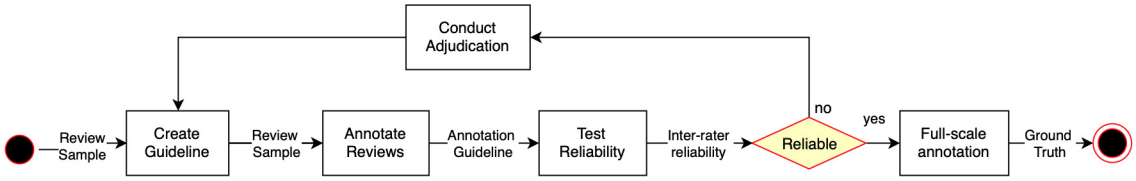


Fig. 2. The Method for Ground Truth Creation.

Table 3
Statistics of the ground truth for 1,000 reviews for 8 subject apps.

		App name								Overall
		Evernote	Facebook	eBay	Netflix	Spotify	Photo editor	Twitter	WhatsApp	
Reviews	No. reviews	125	125	125	125	125	125	125	125	1,000
	Avg. review length	48.30	37.90	32.54	43.46	23.62	12.38	15.79	14.47	28.59
	No. sentences	367	327	294	341	227	154	183	169	2,062
	Avg. sentence length	16.45	14.49	13.84	15.93	13.00	10.05	10.79	10.70	13.85
	Sentence per review	2.94	2.62	2.35	2.73	1.82	1.23	1.46	1.35	2.06
Sentiment	No. sentiments	295	242	206	262	180	96	122	118	1,521
	No. positive	97	49	95	79	32	39	5	20	416
	No. neutral	189	168	102	159	122	47	93	84	964
	No. negative	9	25	9	24	26	10	24	14	141
Features	No. features	295	242	206	262	180	96	122	118	1,521
	No. distinct features	259	204	167	201	145	80	99	100	1,172
	No. single-word features	82	80	78	94	69	39	39	49	530
	No. multi-word features	213	162	128	168	111	57	83	69	991
	Feature per review	2.36	1.94	1.65	2.10	1.44	0.77	0.98	0.94	1,52
Agrmt.	F_1 measure	0.76	0.73	0.77	0.75	0.67	0.78	0.79	0.83	0.76
	Fleiss' Kappa	0.64	0.77	0.77	0.55	0.75	0.86	0.69	0.80	0.73

using the guideline [24]. We evaluated the reliability of their annotation using the inter-rater agreement metrics F_1 and Fleiss' Kappa [39,40]. F_1 is suitable for evaluating text spans' annotations such as feature expressions found in reviews; Fleiss' kappa is suitable to assess inter-rater reliability between two or more coders for categorical items' annotations such as users' sentiment (positive, negative, or neutral). We evaluated inter-rater agreement to ensure the annotation task was understandable, unambiguous, and could be replicated [38]. When disagreement was found, the annotators discussed to adjudicate their differences and refined the annotation guidelines. The process was performed iteratively, each time with a new random sample of reviews until the quality of the annotation was at an acceptable level [39]. Once this was achieved, annotators conducted a full-scale annotation on a new random sample of 1,000 reviews that resulted in our ground truth [24].

(C) Ground truth

Table 3 reports statistics of our ground truth. These statistics concern subject app reviews, annotated opinions (feature-sentiment pairs) and inter-rater reliability measures. The average length of reviews and sentences is measured in words. Statistics of opinions are reported separately for features and sentiments. The number of features has been given for all the annotated features, distinct ones, and with respect to their length (in words). The number of sentiments has been described including their number per polarity.

The ground truth consists of 1,000 reviews for 8 subject apps. In total, 1,521 opinions (i.e., feature-sentiment pairs) have been annotated. Their sentiment distribution is unbalanced: most feature-sentiment pairs are neutral. Among 1,521 annotated features, 1,172 of them are distinct (i.e. mentioned only once).

The feature distribution in app reviews can be found in Fig. 3(a). A large number of reviews do not refer to any specific feature. 75% of reviews refers to no feature or to only one or two features. Fig. 3(b) provides the feature length distribution. The median length for a feature is 2 words, 75% of features has between 1 and 3 words, and nearly 5% has more than 5 words.

4.1.3. Evaluation metrics

We used precision and recall metrics [39] to answer RQ1 and RQ2. We used them because feature extraction is an instance of information extraction problem [39], whereas sentiment analysis can be seen as a classification problem [16].

(A) Evaluation Metrics For Feature Extraction

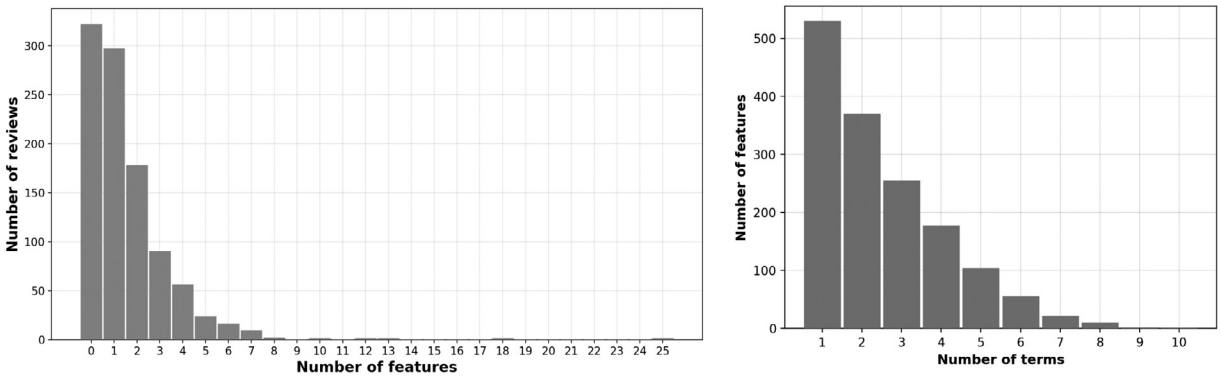
In answering RQ1, precision indicates the percentage of extracted features that are true positives. Recall refers to the percentage of annotated features that were extracted. An extracted feature can be true or false positive. True positive features correspond to features that were both extracted and annotated; False positives are features that were extracted but not annotated; Annotated but not extracted features are called false negative. To determine whether an extracted feature is true or false positive, we compared them with annotated features in the ground truth. To this end, we used the following *feature matching method*:

Let Γ be the set of words in a review sentence and $f_i \subseteq \Gamma$ be the set of words used to refer to feature i in that sentence. Two features $f_1, f_2 \subseteq \Gamma$ match at level n (with $n \in \mathbb{N}$) if and only if (i) one of the feature is equal to or is a subset of the other, i.e. $f_1 \subseteq f_2$ or $f_2 \subseteq f_1$, and (ii) the absolute length difference between the features is at most n, i.e. $\|f_1| - |f_2\| \leq n$.

B) Evaluation Metrics For Feature-Specific Sentiment Analysis

In answering RQ2, precision indicates the percentage of predicted sentiments that are correct. Recall refers to the percentage of annotated sentiments that are predicted correctly. To determine whether predicted sentiments are correct, we compared them with annotated ones in the ground truth.

We measured precision and recall for each polarity category (i.e. positive, neutral and negative). We also calculated the overall



(a) Feature distribution in app reviews. (b) Distribution of feature length.

Fig. 3. Feature distribution in app reviews, and Feature length distribution.

Table 4
RQ1. Results for feature extraction at varied levels of feature matching.

App name	Exact match (n=0)						Partial match 1 (n=1)						Partial match 2 (n=2)					
	GuMa		SAFE		ReUS		GuMa		SAFE		ReUS		GuMa		SAFE		ReUS	
	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
Evernote	0.06	**0.13**	**0.07**	0.08	**0.07**	0.08	0.15	**0.35**	**0.22**	0.24	0.19	0.20	0.17	**0.39**	**0.32**	0.35	0.27	0.29
Facebook	0.03	0.07	0.03	0.03	**0.09**	**0.09**	0.10	**0.28**	**0.15**	0.17	**0.15**	0.14	0.13	**0.36**	**0.23**	0.26	0.20	0.19
eBay	0.04	**0.07**	0.04	0.05	**0.06**	0.06	0.14	**0.26**	**0.22**	**0.26**	0.14	0.14	0.17	0.32	**0.34**	**0.39**	0.22	0.21
Netflix	0.03	**0.13**	0.03	0.03	**0.06**	0.07	0.11	**0.45**	**0.19**	0.21	0.18	0.21	0.13	**0.55**	**0.27**	0.29	0.25	0.29
Spotify	0.05	0.10	0.05	0.04	**0.15**	**0.13**	0.18	**0.37**	**0.24**	0.23	0.23	0.20	0.21	**0.43**	**0.36**	0.34	0.29	0.26
Photo Editor	0.12	0.11	0.12	0.09	**0.14**	**0.13**	0.26	0.25	**0.34**	**0.27**	0.23	0.21	0.29	0.27	**0.38**	**0.30**	0.27	0.25
Twitter	**0.06**	**0.19**	**0.06**	0.07	0.02	0.02	0.16	**0.49**	**0.23**	0.24	0.11	0.11	0.18	**0.58**	**0.35**	0.36	0.27	0.26
WhatsApp	0.05	**0.21**	**0.11**	0.11	0.06	0.06	0.14	**0.56**	**0.32**	0.33	0.19	0.20	0.16	**0.64**	**0.39**	0.40	0.24	0.25
Mean	0.05	**0.13**	0.06	0.06	**0.08**	0.08	0.15	**0.37**	**0.24**	0.24	0.18	0.18	0.18	**0.44**	**0.33**	0.34	0.25	0.25

precision and recall of all three sentiment polarities. To this end, we used the weighted average of precision and recall of each polarity category. The weight of a given polarity category was determined by the number of annotated sentiments with the sentiment polarity.

4.2. Results

RQ1: What is the effectiveness of feature extraction approaches?

To answer RQ1, we compared extracted features to our ground truth using *feature matching method* at levels 0, 1 and 2 (see Section 4.1.3). We selected these levels as extracted and annotated features may differ by a few words but still indicating the same app feature. We then computed precision and recall at these levels. Table 4 reports precision and recall for each approach at different matching levels (best in bold). The results show the approaches achieved low precision, recall given Exact Match. For all three approaches, precision and recall increase when we loosen the matching criteria to partial matching with $n = 1$ or 2. The growth can be attributed to the changed numbers of true positives (TPs), false positives (FPs) and false negatives (FNs) when n increases. Fig. 4 shows their behavior as the matching level n increases; $\Delta TPs = -\Delta FPs = -\Delta FNs$ when n increases.

RQ2: What is the effectiveness of feature-specific sentiment analysis approaches?

In answering RQ2, we report the effectiveness of ReUS and GuMa in feature-specific sentiment (see Section 4.1.3). To this end, we compared predicted and annotated sentiments, and exploited a subset of the ground truth with opinions (feature-sentiment pairs) we used to answer RQ1. Indeed, since ReUS predicts sentiments only for extracted features, we considered only true positive features obtained in answering RQ1 and formed three datasets, each corresponding to true positive features (and their sentiment) from Exact Match, Partial Match$_1$ and Partial Match$_2$. Table 5 reports for each dataset the total number of opinions, and their breakdown to polarity categories. We also evaluated GuMa with these datasets and with all the annotated opinions in our ground truth.

Table 5
RQ2. Dataset used for evaluating feature-specific sentiment analysis.

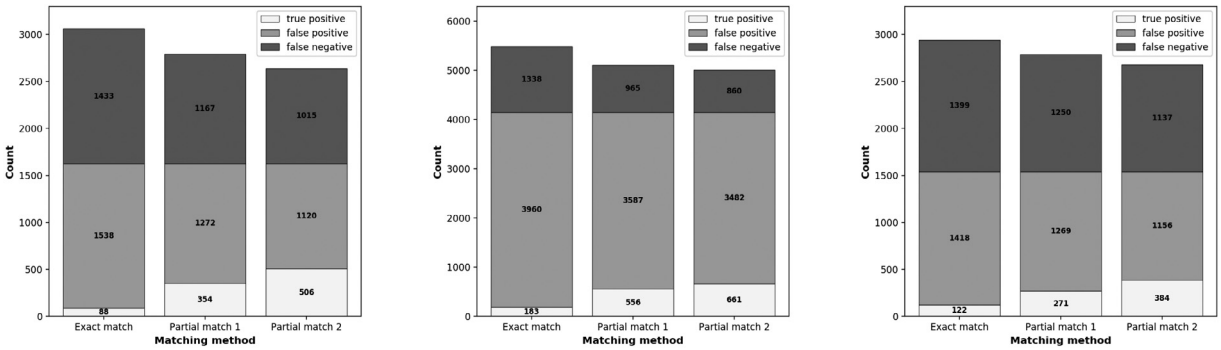
Dataset	# Opinions	# Positive	# Neutral	# Negative
Exact match	122	56	52	14
Partial match 1	271	97	149	25
Partial match 2	384	120	226	38
All annotated	1521	416	964	141

Table 6
RQ2. Results for feature-specific sentiment analysis (overall).

Dataset	Approach	# Correct prediction	Precision	Recall
Exact match	ReUS	**85**	**0.74**	**0.70**
	GuMa	77	0.65	0.63
Partial match 1	ReUS	**184**	0.69	**0.68**
	GuMa	176	**0.72**	0.65
Partial match 2	ReUS	**265**	0.69	**0.69**
	GuMa	252	**0.73**	0.66
All annotated	ReUS	–	–	–
	GuMa	**958**	**0.73**	**0.63**

The answer to RQ2 can be given at two levels of details, the overall effectiveness of predicting a sentiment, and the effectiveness of predicting a specific polarity (e.g., positive). We report our results at both levels of details.

Overall effectiveness. Table 6 reports the number of correct predictions, and weighted precision/recall for inferring overall sentiment (best in bold). We can observe that ReUS achieves higher



(a) SAFE (b) GuMa (c) ReUS

Fig. 4. RQ1. No. TPs, FPs and FNs as the level of features matching changes.

Table 7
RQ2. Results for feature-specific sentiment analysis (per each polarity).

Dataset	Approach	Positive			Neutral			Negative		
		# Correct prediction	Precision	Recall	# Correct prediction	Precision	Recall	# Correct prediction	Precision	Recall
Exact match	ReUS	35	**0.90**	0.62	**45**	0.60	**0.87**	5	**0.62**	0.36
	GuMa	**47**	0.68	**0.84**	21	**0.68**	0.40	**9**	0.41	**0.64**
Partial match 1	ReUS	47	**0.80**	0.48	**131**	0.66	**0.88**	6	**0.43**	0.24
	GuMa	**86**	0.61	**0.89**	73	**0.85**	0.49	**17**	0.40	**0.68**
Partial match 2	ReUS	53	**0.80**	0.44	**205**	0.68	**0.91**	7	**0.41**	0.18
	GuMa	**107**	0.59	**0.89**	122	**0.86**	0.54	**23**	0.38	**0.61**
All annotated	ReUS	–	–	–	–	–	–	–	–	–
	GuMa	**355**	**0.52**	**0.85**	**510**	**0.87**	**0.53**	**93**	**0.36**	**0.66**

precision and recall than GuMa for Exact Match dataset, whereas both approaches have similar performances on the Partial Match_1 and Partial Match_2 datasets.

Specific effectiveness. In Table 7, we report the metrics showing the effectiveness of the approaches in predicting specific polarities (best in bold). The results show that on positive opinions ReUS achieves higher precision while suffering from lower recall. Conversely, on neutral opinions GuMa provides better precision but lower recall than ReUS. When looking at the approaches, the analysis of the results revealed that none of the approaches was able to reliably assess the sentiment of negative options. Both approaches were good at discriminating between positive and negative opinions. Most incorrect predictions were caused by misclassifying positive/negative sentiment with neutral one and vice versa.

4.3. Discussion

The results indicate that the approaches have limited effectiveness in mining user opinions. Our findings bring into question their practical applications.

(A) Feature Extraction

In our experiment, feature extraction methods have lower precision and recall than previously reported [15,17,18]. SAFE was reported with 0.71 recall [15]. Our results show the approach achieves 0.34 recall for the least rigorous evaluation strategy which can be attributed to several factors; SAFE uses dataset-specific linguistics patterns to identify features. These patterns have been defined in the original study [15]. The characteristic of our dataset is different than one used in the original evaluation in terms of apps we used, and the vocabulary of their reviews [6,41]. We observed the tool frequently identifies a part of a feature expression (e.g., "organize list") instead of detecting a complete one (e.g., "keeping me organized with my lists"); and it does not identify features expressed using a single word (e.g., "synchronization"). This leads to an increased number of false positive and false negative features, lowering both precision and recall. We also observed SAFE does not handle co-reference resolution; and therefore it cannot detect features to which users refer using pronouns [32].

The majority of features extracted by GuMa are incorrect. Although GuMa initially reported precision and recall of 0.58 and 0.52 [17], our experiment found lower figures of 0.18 precision and 0.44 recall. Although the difference may be due to our re-implementation of the GuMa method, we have taken great care in re-implementing the method as described in the paper as rigorously as possible. We tested that our implementation is a consistent approximation of the original tool and produces the same outputs based on the examples reported in the paper. Unfortunately, the original GuMa implementation was not available for comparison. We however argue the imprecision of GuMa comes from the tool's feature extraction algorithm. The collocation finding algorithm suffers from an increased number of false positives as it frequently extracts collocations that are not app features, but simply commonly mentioned expressions (e.g., "easy use"); and it consequently decreases the precision. On the other hand, the algorithm only extracts frequently discussed expressions; GuMa thus is not able to detect features that are not popularly discussed (e.g., mentioned once). Future works can extend the tool with linguistic patterns describing the language structure in which users express features. It would enable the tool to identify non-common features using such patterns in addition to frequently appearing ones.

We believe ReUS suffered from low precision and recall because it was designed to extract features from product reviews in an online commerce website (Amazon) rather than from app reviews in app stores [16]. Not only the characteristics of product reviews can be different than app reviews [16], but also the feature definition of non-software products may be varied [18].

For example, app features are commonly expressed as a combination of verb and noun (e.g., "send messages"). However, ReUS only identifies features expressed as a noun or consecutive nouns (e.g., "video camera"); moreover, such expressions must be in a syntactic dependency relation with an opinion word like adjective or adverb (e.g. "video camera is great"); otherwise, features are not identified. App users rarely discuss features in a such way; and thus, the tool does not perform well.

More importantly, our findings support a conjecture that the original evaluation procedures of SAFE and GuMa led to over-optimistic results. The limitations of these procedures have been questioned recently [21,22]. These procedures did not define a feature matching strategy [22], relied on a subjective judgment [15,21], and used a biased dataset [17,21,22]. We hope our new annotated dataset and description of our evaluation method will contribute to improving the quality of feature extraction techniques and their evaluations.

(B) Feature-Specific Sentiment Analysis

Our investigation of results (RQ2) concludes that the overall effectiveness of the approaches is promising (see Table 6). However, it reveals that their precision and recall differ considerably by sentiment class (positive, negative, or neutral). The approaches provide satisfactory performance for predicting positive and neutral sentiments. But they suffer from inaccurate predictions for negative sentiments. Like previous studies [42,43], we observed positive and neutral opinions are expressed more directly and explicitly than negative ones. Users tend to express negative opinions indirectly (e.g., "there are big time gaps between the tweets that do load"), sarcastically (e.g., "search takes an eternity") and informally (e.g., "WTH....going on, I cannot see the photos"). Moreover, negative opinions are commonly expressed using jargon that is specific for the software domain (e.g., "bug", "crash", or "please fix!"); Both ReUS and GuMa cannot handle it. Their lexical dictionaries therefore should be expanded with such vocabulary to enhance the sentiment analysis performance.

Overall, we are surprised by the comparable effectiveness of both approaches. We expected ReUS to outperform GuMa. ReUS exploits a sophisticated technique to detect an opinion word in a sentence that carries a feature-specific sentiment; GuMa makes predictions based on a simplified premise that a feature-specific sentiment corresponds to the overall sentiment of a sentence.

(C) Implication on Requirement Engineering Practices

Identifying what precision and recall app review mining techniques should have to be useful for requirements engineers in practice is an important open question [44]. In principle, a tool facilitating opinion mining should synthesize reviews so that the effort for their further manual analysis would be negligible or at least manageable. Clearly, this effort depends on a scenario the approach intends to support. Given a scenario of prioritizing problematic features, a developer may seek for information about the number of specific features that received negative comments, for example to understand their relevance. To this end, both information about extracted features and their predicted sentiments should be accurate and complete. Our results, however, show that feature extraction techniques generate many false positives. Given the large number of extracted features, filtering out false positives manually may not be cost-effective. We may imagine that the problem could be partially addressed using a searching tool [13]; Requirements engineers could use the tool to filter out uninteresting features (including false positives) and focus on those of their interest.

However, other issues remain unsolved. Feature extraction techniques fail to identify many references to features (they have low recall), and sentiment analysis techniques perform poorly for identifying feature-specific negative sentiments.

4.4. Threats to validity

Internal Validity. The main threat is that the annotation of reviews was done manually with a certain level of subjectivity and reliability. To overcome the risk we followed a systematic procedure to create our ground truth. We prepared an annotation guideline with definitions and examples. We conducted several trial runs followed by resolutions of any conflicts. Finally, we evaluated the quality of the annotation using inter-rater agreement metrics.

External Validity. To mitigate the threat, we selected reviews for popular apps belonging to different categories and various app stores. These reviews are written using varied vocabulary. We, however, admit that the eight apps in our study represent a tiny proportion of all the apps in the app market. Although our dataset is comparable in size to datasets in previous studies [15,17,18], we are also exposed to sampling bias.

Construct Validity. The main threat is the extent to which our operationalization of a feature, a sentiment and a user opinion reflects the actual constructs under study [45]. To mitigate the threat, we first defined their conceptual meaning referring to the requirement engineering and opinion mining literature [16,46]. We then chose standard variables to represent each concept in a text document: a bounded textual phrase referring to a mentioned feature; the polarity of a review sentence where the phrase appears referring to the user's associated sentiment; and their pair referring to a user opinion. To ensure the conceptual meaning and the operational definitions are understandable, we discussed them with an independent panel of researchers, including experts in requirements engineering and natural language processing. To verify the reliability of the operationalization, we tasked two human-coders to annotate a sample of app reviews using the operational definitions, and then checked their inter-rater agreement is of sufficient quality.

5. Study II: Searching for feature-related reviews

The second empirical study evaluated and compared the approaches addressing the problem of searching for feature-related reviews (see Section 2.1). We here describe the design of the study, report and discuss the results, together with threats to validity of this second study.

5.1. Empirical study design

In this section, the research question we aimed to answer is presented, together with the collected and manually annotated dataset, the evaluation procedure as well as evaluation metrics we used in answering the question.

5.1.1. Research questions

The objective of the study was to evaluate and compare approaches searching for feature-related app reviews. To this end, we formulated the following research question:

RQ3: What is the effectiveness of approaches in searching for app reviews pertinent to a particular feature?

For RQ3, we investigated the degree to which the selected approaches can correctly search for app reviews pertinent to a particular feature. A conclusive method of measuring the correctness of searching for feature-related reviews is by relying on a human judgment; we first tasked human-coders to formulate a set of queries (i.e., app features) based on descriptions of selected apps. We next fed the subject approaches with these queries and a set of reviews. The human coders then evaluated the top-n reviews retrieved by the approaches for the queries. In answering RQ3, we report precision@n, average precision and relative recall for each approach.

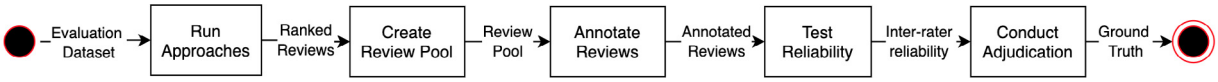


Fig. 5. The Method for Review Pool Creation.

Table 8
The overview of the selected apps and collected reviews.

App Name	Category	Platform	#Reviews
Evernote	Productivity	Amazon	1,250
Facebook	Social	Amazon	1,250
eBay	Shopping	Amazon	1,250
Netflix	Movies & TV	Amazon	1,250
Spotify music	Audio & Music	Google Play	1,250
Photo editor pro	Photography	Google Play	1,250
Twitter	News & Magazines	Google Play	1,250
Whatsapp	Communication	Google Play	1,250

5.1.2. Manually annotated dataset

To create the dataset we used to answer RQ3, we collected reviews for selected apps presented in the first study as well as their app descriptions. We then asked human-coders to identify queries (i.e., app features) from these descriptions. We next fed the approaches with these queries; then tasked the coders to annotate a samples of retrieved reviews with respect to the queries. The resulting dataset is publicly available [24].

(A) Collected Dataset

We used the same dataset we had collected in the first study, including reviews and descriptions for 8 popular apps from different categories (see Table 2). We then randomly sampled 1,250 reviews per app (in total 10,000 reviews) as we strove to obtain a representative review sample. Table 8 reports the summary of our dataset, indicating the apps and the size of review samples we used.

(B) Query Set

We built a query set based on the collected app descriptions conveying information about an app and their features. We first tasked two human-coders[3] to identify app features from the descriptions following a systematic procedure analogous to the one used in the first experiment for feature identification from app reviews (see Section 4.1.2). Since human-coders had experience with an annotation process and a common understanding of 'app feature' from the first study (see Section 4), they conducted the feature annotation without pre-training. We however validated their inter-rater agreement to ensure their annotation was reliable and sufficient quality [38]. We evaluated the inter-rater using agreement F_1 metric as it is suitable for evaluating text spans' annotations such as feature expressions found in app descriptions [39,40]. Table 9 reports the statistics of the subject app descriptions, annotated features and inter-rater reliability measures. The complete list of the identified features can be found in the supplementary materials [24]. In total, 124 app features (candidate queries) have been annotated in the app descriptions. The average inter-rater agreement between coders was 0.78, indicating the substantial reliability of the annotation [39]. The length of the identified features ranges between 1 and 5 words, with the average of 2 words.

To form the evaluation query set for our experiment, we randomly selected 3 features for each app (in total 24 features). We selected this number as we wanted to obtain a broad and diverse set of queries for our evaluation [40]. Table 10 shows details of these queries. We used this query set in the review annotation procedure. We fed the approaches with these queries, then the human-coders assessed their outcomes.

[3] We tasked the same human-coders who had annotated the dataset in the first study.

(C) Review Annotation Procedure

We used the pooling method to evaluate the performance of the selected approaches searching for feature-related reviews [40]. We opted for this method as it is commonly used by researchers for evaluating approaches addressing the information retrieval problem [40,47]; in particular, when the assessment of their results is limited due to the large size of evaluation dataset. The problem of searching for feature-related reviews can be seen as an instance of the general information retrieval problem.

In this method, the top-n reviews (with n = 20) from the rankings obtained by the evaluated approaches are merged into a pool, duplicates are removed, and the reviews are presented in a random order to human-coders annotating their relevance with regards to input queries. We selected the top-20 reviews as this level of the pool's depth is recommended in the information retrieval literature [47]; it reduces the number of documents that human-coders need to annotate and enables to calculate stable values of evaluation metrics. Fig. 5 illustrates the overview of the procedure. We first inputted the selected approaches with a query and a review sample. We then obtained ranked results of top-n reviews from each evaluated approach (n=20 in our study). We merged their results into a pool of unique reviews and removed duplicates. The review pool was next presented in a random order to the coders who annotated it with the query set. Each assessor judged the relevance of the reviews with respect to an input query; a review was classified as relevant if it refers to the queried feature, and irrelevant otherwise. We measured their inter-rater agreement to assess the task was understandable, unambiguous, and could be replicated. We employed Fleiss' Kappa to this end as it is suitable for evaluating inter-rater reliability between two or more coders for categorical items' annotations such as a review's relevance [39,40]. The quality of the annotation was always at least at substantial level [39]. Therefore, the coders discussed the minor differences in their annotations, adjudicated them and provided an annotated dataset, comprising of a query and a pool of respectively annotated reviews. We repeated the method for each query and corresponding new review sample. Having the annotated reviews for all the queries, we obtained the ground truth that we used to assess the performance of the approaches. Table 11 reports the statistics of the ground truth. These statistics concern annotated reviews and inter-rater reliability measures. The number of reviews is reported in relation to a concrete query, indicating relevant reviews (query-related) and non-relevant (remaining). In total, 1,113 reviews have been annotated with respect to 24 queries. Among 1,113 annotated reviews, 512 of them are relevant and 601 are non-relevant. On average, 46 reviews have been annotated per a query. The number of relevant reviews ranges between 2 and 54, with the mean of 21 reviews; whereas the number of non-relevant reviews pear query is between 4 and 49, with the mean of 25 reviews. The inter-rater agreement indicates the substantial reliability of the dataset [39,40].

5.1.3. Evaluation metrics

We used precision@n, average precision and relative recall metrics [40,47,48] to answer RQ3. We used them because searching for feature-related reviews is an instance of information retrieval problem [40]. Precision@n indicates the percentage of the top-n retrieved reviews that are relevant [47]. The metric is useful for assessing the searching task in finding the most relevant

Table 9
Statistics of the identified features from app descriptions for 8 subject apps.

		App name								Overall
		Evernote	Facebook	eBay	Netflix	Spotify	Photo editor	Twitter	WhatsApp	
Description	No. words	500	72	424	264	144	195	269	355	2,223
	No. sentences	33	8	39	18	14	21	28	20	181
	Avg. sentence length	15.15	9.00	10.87	14.67	10.29	9.29	9.61	17.75	12.08
	No. paragraphs	8	2	13	6	6	7	9	12	63
Features	No. features	18	7	24	10	10	19	21	15	124
	Min. feature length	1	1	2	2	2	1	1	1	1
	Max. feature length	4	4	5	3	5	4	4	5	5
	Avg. feature length	2.06	2.14	2.50	2.50	3.00	2.11	2.33	2.40	2.07
	No. single-word features	4	2	0	0	0	3	3	1	13
	No. multi-word features	14	5	24	10	10	16	18	14	111
Agr.	Fleiss' Kappa	0.82	0.88	0.75	0.68	0.67	0.84	0.70	0.89	0.78

Table 10
The set of query for the empirical evaluation.

App name	Id	Query	App name	Id	Query
Evernote	1	Create shortcuts	Spotify	13	Play playlist on shuffle mode
	2	Write notes		14	Create playlist of songs
	3	Annotate documents		15	Offline listening
Facebook	4	Chat	Photo Editor	16	Edit a photo
	5	Share		17	Photo filters
	6	Watch videos		18	Build photo collage
eBay	7	Bid item	Twitter	19	Twitter Moment
	8	Search for offer on item		20	Follow people on Twitter
	9	List items for sale		21	Write a Tweet
Netflix	10	Rate movies	WhatsApp	22	Notifications
	11	Search for titles		22	WhatsApp call
	12	Watch movies		24	Send messages

Table 11
Statistics of the ground truth, indicating no. reviews in relation to concrete queries.

	App name												
	Evernote			Facebook			eBay			Netflix			
Query Id	1	2	3	4	5	6	7	8	9	10	11	12	
No. reviews	47	51	45	46	45	36	48	48	48	55	42	58	
No. relevant	2	40	4	23	24	31	39	25	21	6	16	54	
No. non-relevant	45	11	41	23	21	5	9	23	27	49	26	4	
Fleiss' Kappa	1.00	0.84	1.00	1.00	1.00	1.00	1.00	0.92	1.00	1.00	0.88	0.88	
	App name												Overall
	Spotify			Photo Editor			Twitter			WhatsApp			
Query Id	13	14	15	16	17	18	19	20	21	22	23	24	24
No. reviews	46	43	47	40	45	48	47	47	49	39	48	45	1,113
No. relevant	21	13	16	31	15	9	2	18	15	13	43	31	512
No. non-relevant	25	30	31	9	30	39	45	29	34	26	5	14	601
Fleiss' Kappa	1.00	1.00	1.00	1.00	1.00	1.00	0.66	1.00	0.95	0.89	1.00	0.83	0.95

documents at a given rank. Average precision summarizes the ranking of top-n retrieved reviews by averaging the precision@n values from the rank positions where a relevant review was retrieved [40]. The metric is based on the ranking of all the relevant reviews, but their value depends heavily on the highly ranked relevant reviews. It is thus a suitable measure for evaluating the task of as many relevant reviews as possible while still reflecting the intuition that the top-ranked reviews are the most important. Relative recall is the proportion of known relevant reviews that has been retrieved in top-n results [48]. The metrics provide partial knowledge about the completeness of retrieved results. To determine whether retrieved reviews are relevant, we compared them with the annotated ones in the ground truth. We also used it to determine the known relevant reviews.

5.2. Results

RQ3: What is the effectiveness of approaches in searching for app reviews pertinent to a particular feature?

In answering RQ3, we report the effectiveness of Lucene, MARAM and SAFE in searching for feature-related reviews. To this end, we compared the top-20 retrieved reviews to our ground truth. We selected this level of depth as it is typically recommended in the literature [40,48]. Table 12 reports precision@20 (P@20), average precision (AP) and relative recall (RR) for each approach (best in bold). The results show the approaches achieved substantially different performance among each other. Lucene scored the best results given all three metrics, whereas MARAM

Table 12
RQ3. Results of the evaluated approaches searching for feature-related reviews.

App name	Lucene			MARAM			SAFE		
	P@20	AP	RR	P@20	AP	RR	P@20	AP	RR
Evernote	**0.42**	**0.83**	**0.67**	0.32	0.41	0.47	0.25	0.61	0.36
Facebook	**0.97**	**0.99**	**0.75**	0.85	0.98	0.66	0.32	0.51	0.22
eBay	0.73	0.82	0.52	**0.82**	**0.90**	**0.59**	0.37	0.62	0.25
Netflix	**0.57**	0.70	**0.62**	0.47	**0.82**	0.32	0.37	0.74	0.19
Spotify	**0.58**	**0.85**	**0.70**	0.42	0.45	0.49	0.23	0.32	0.29
Photo editor	**0.68**	**0.99**	**0.80**	0.33	0.68	0.24	0.38	0.27	0.38
Twitter	**0.47**	0.66	**0.70**	0.32	**0.77**	0.39	0.12	0.44	0.29
WhatsApp	**0.85**	**0.99**	**0.68**	0.70	0.94	0.56	0.53	0.75	0.33
Mean	**0.66**	**0.85**	**0.68**	0.53	0.74	0.46	0.32	0.53	0.29

achieved the second best results. Both Lucene and MARAM approaches achieved promising precision in returning relevant results among the top-20 retrieved reviews. While looking at their average precision, we can also observe the approaches showed good effectiveness in ranking most relevant reviews in the top of their results. Whereas, relative recall suggests the approaches retrieved a substantial portion of the relevant reviews. Conversely, we observe that SAFE achieved the lowest performance given all three metrics. On average, one-third of the reviews returned by SAFE is relevant to queried features; the average precision indicates these reviews were scattered over the ranked results. In our experiment, SAFE's relative recall ranges from 19% to 38%. This result can be attributed to the poor effectiveness of the tool in performing feature extraction (see Section 4.2); the outcome of this task affects the input of the actual task of searching for feature-related reviews (see Section 2).

5.3. Discussion

The results shed a new light on the effectiveness of the evaluated approaches. Our findings suggest the approaches can be useful for practical applications.

(A) Searching For Feature-Related Reviews

In our experiment, the three approaches achieved diverse effectiveness (see Table 12). The results suggest SAFE achieves lower performance than previously reported. The approach was originally reported with 0.70 precision and 0.56 recall [15]. Our results show the approach achieves 0.32 precision@20, 0.53 average precision and 0.29 recall rate. We argue the discrepancy between the original results and ours cannot be attributed to the evaluation metrics. We observed the tool exploits a simplistic rule and highly ranks reviews in which a single word of a query appears; this leads to retrieving many false positive reviews and decreased precision. In addition, the original evaluation of the tool was limited to a fraction of app reviews from which the tool could first correctly extract features. Such a condition simplifies the overall evaluation procedure, but it also eliminates possible matches between queried features and misidentified ones from reviews. We therefore hypothesize the original evaluation procedure led to over-optimistic results. The limitations of the procedure were questioned recently [19,21,22]. The procedures relied on a subjective judgment [15,21], biased and small evaluation dataset [17,21,22]. We have taken great care to overcome the limitations in our study; we employed much larger and diverse dataset; then followed a rigorous evaluation procedure. We thus argue the strength of evidence is generally larger in our study.

Our findings shed a new light on the usefulness of MARAM and Lucene. Though they have never been empirically evaluated for searching for feature-related reviews [7], the approaches achieved much better performance than SAFE. We took great caution in re-implementing MARAM as described in the original study as rigorously as possible; we tested that our approximation of the tool behaves similarly to the original one using examples in the original study. Similarly, we implemented Lucene following their documentations to make the most of it [23]. The most striking result to emerge from the data, however, is that Lucene, a standard search engine library, achieved the best effectiveness; No previous study tried to exploit the tool to searching for feature-related reviews, but rather focused on developing new techniques. Importantly, the tool found relevant reviews within seconds. In contrast, it took between several dozen minutes up to several hours for the other tools. In our opinion these results provide useful evidence to researchers aiming at developing new mining techniques about the opportunity to exploit existing searching techniques; and pay more attention to the efficiency of their approaches. We believe our new annotated dataset and the evaluation procedure will help to improve the quality of these techniques evaluation.

(B) Implication on Requirement Engineering Practices

The results suggest Lucene tool can be useful for requirements engineering use cases. Supposing requirements engineers have been tasked to detail requirements about concrete features, the tool could help them to quickly identify what users have been saying about the features. Though the engineers would need to filter-out a few unrelated reviews, the results indicate most of them, in particular, those highly ranked one would be of their interest. This cheap elicitation technique might be not sufficient in itself, but can be of great value when used in combination with other techniques e.g., for classifying user feedback [49]. Integrating the techniques could help the engineers to further distill reviews reporting problems about these features, requesting improvements or discussing their quality attribute. As for the usefulness of the tool for requirements prioritization, the engineers would seek for the information about the concrete number of feature-related comments that are negative or report a certain type of requests. Future studies thus should investigate the tool's performance for outputting the set of reviews. It would require studying the cut-off value of the tool's similarity measure to discriminate feature-related reviews from unrelated ones.

5.4. Threats to validity

Internal Validity. The main threat is that the annotation of both: app description and reviews was done manually with a certain level of subjectivity and reliability. To mitigate the threat, we employed a systematic procedure to create our evaluation dataset. We prepared an annotation guideline with definitions and examples. We then evaluated the quality of the annotation task using standard inter-rater agreement metrics [39,40]. Another limitation concerns the lack of evaluation of 'the absolute' precision and recall. We only estimated their 'relative' values using the annotated sample of app reviews that the tools had outputted; we did not annotate the complete set of collected app reviews to prepare the evaluation dataset. Identifying relevant app reviews to a particular query from a collection of thousands

Table 13
The differences between our study and previous evaluations for RQ1 and RQ2.

	Criterion	Our study	SAFE [15]	GuMa [17]	ReUS [18]
Evaluation	No. Approaches	**3**	2	1	1
	Feature Extraction	**Yes**	**Yes**	No	**Yes**
	Sentiment Analysis	**Yes**	–	**Yes**	**Yes**
	Method	**Automatic**	Manual	Manual	**Automatic**
Ground truth	Released	**Yes**	No	No	No
	No. Apps	**8**	5	7	–
	No. Reviews	1000	80	**2800**	1000
	No. App Stores	**2**	1	**2**	–
	Dataset Analysis	**Yes**	No	No	**Yes**

or millions of app reviews, like in our dataset, is a non-trivial task. A single human-coder, for instance, would require 83 h to judge the collection of 10,000 reviews for a single query (30 sec/review). Precision and recall can thus be estimated only. We used the standard pooling method to prepare our annotated dataset and estimate the metrics [47]. We admit the use of this method cannot provide the complete knowledge about 'the absolute' precision and recall [40]; and their actual values remain unknown. Estimating their 'relative' values is however still useful; it helps to benchmark the tools and to obtain the partial knowledge about their performance.

External Validity. To mitigate the threat, we selected apps belonging to diverse categories and different app stores. Their descriptions as well as user feedback has diverse characteristics: different length, varied vocabulary and refereed to different app features. We, however, admit that our dataset comprises a tiny fraction of all the apps in the app market. Though its size is much greater compared to previous studies [13,15], we are also exposed to sampling bias [50].

Construct Validity. A potential threat to our study is the extent to which our feature operationalization reflects the actual construct under study [45]. To reduce this threat, we defined the conceptual meaning of the feature referring to the requirement engineering literature [16,46]; we then chose the standard variable representing the concept in a textual document: a bounded textual phrase referring to a feature mentioned in an app review. We further confirmed the meaning and the operationalization are understandable to external requirements engineering and natural language processing scholars. To ensure the operationalization was reliable, we checked the inter-rater agreement of human-coders annotating sample reviews with the concept was of sufficient quality.

6. Related works

We focus the discussion of related work in the light of previous studies on (i) Mining User Opinions and (ii) Searching For Feature-Related Reviews.

6.1. Mining user opinions

Previous works have proposed benchmarks for app review analytics (e.g. [22,42]) but with objective different than ours. Table 13 shows the differences between our study and previous works, pointing out the different criteria that guided the evaluations, which are grouped into Evaluation and Ground Truth categories. The first includes criteria such as the number of evaluated approaches, evaluated tasks and a method type used for their evaluation. The latter includes characteristics of datasets.

In our study, we evaluated three approaches: SAFE, GuMa and ReUS. We assessed them in addressing problems of feature extraction and sentiment analysis. Johann et al. [15] also compared SAFE to GuMa [17]. Our study extends their evaluation by including ReUS [18]. Unlike the original study [17], we evaluated GuMa in performing a feature extraction rather than modeling feature topics. We also compared the approach to ReUS in inferring a feature-specific sentiment.

We used a different methodology for evaluating SAFE and GuMa [15,17]; The correctness of their solutions has been evaluated manually [15,17]. The judgment criteria, however, has not been defined. Such a procedure suffered from several threats to validity such as human error, authors' bias and the lack of repeatability [21]. To address the limitations, we adopted automatic matching methods and defined explicit matching criteria. The ground truth in our study differs from that used in previous works. Unlike Dragoni et al. [18], we evaluated ReUS using app reviews. The authors used a dataset composed of comments for restaurant and laptops. As Johann [15] and Guzman [17], we created an annotated dataset for the evaluation. We, however, used a systematic procedure and assessed the quality of ground truth using acknowledged measures [16,38]. Previous studies did not report a systematic annotation procedure [15] nor measured the quality of their annotation [17]. Their datasets were not analyzed nor made public [15,17].

6.2. Searching for feature-related reviews

More than 182 papers in the area of app review analysis have been published in the last decade [7], but only three of them investigated the use of techniques for task of searching for feature-related reviews [13–15]. Previous works however had different objective than in this study: they focused on developing new approaches to searching for feature-related reviews, whereas this paper focuses on a more extensive evaluation of these approaches and their comparison to a general purpose searching technique [23]. We here discuss the difference between our study and related works based on aspects concerning an empirical evaluation. Table 14 shows the differences between our study and the previous works, pointing out the different criteria guiding this discussion.

Our empirical study evaluated and compared the effectiveness of three approaches: SAFE [15], Lucene [23] and MARAM [14]. The related works focused only on proposing and/or evaluating their searching approaches without bench-marking them with the existing techniques (e.g., [23]). Similarly like SAFE's authors, we also released our data-set publicly. Most importantly, we elaborated and facilitated the first dataset, consisting of annotated reviews with queries. Previous studies either evaluated their approach without such dataset [15], or have not facilitated it for the public scrutiny [13]. The scale of the empirical evaluation also favors our study; in particular in terms of the number of app reviews; In our previous work [13] we elaborated only 200 reviews for a single app with 20 exemplary queries.

7. Conclusion

Mining feature-specific information from app reviews can be useful to guide requirement engineering activities such as user

Table 14
Differences between our study and previous empirical evaluations for RQ3.

Criterion	Our study	SAFE [15]	MARAM [14]	JaDa[a] [13]
No. Approaches	**3**	1	0	1
Validation	**Yes**	**Yes**	No	**Yes**
Dataset Released	**Yes**	**Yes**	–	No
Ground Truth	**Yes**	No	–	No
Np. Apps	**8**	5	–	1
No. Reviews	**1,113**	–	–	200
No. Queries	20	**178**	–	20

[a]We refer to the approach using abbreviations derived from the first author's name (Jacek Dąbrowski).

validation [3,5,6,9], requirements elicitation [3,7,9], or requirement prioritization [3,4]. However, the performance of app review mining techniques and their ability to support these tasks in practice are still unknown [7]. We have presented two empirical studies aimed at evaluating techniques for opinion mining and searching for feature-related reviews.

In the first study, we have evaluated three approaches supporting opinion mining task: SAFE [15] relying on part-of-speech parsing, GuMa [17] adopting a collocation-based algorithm, and ReUS [18] exploiting a syntactic dependency-based parser. We have created a new dataset of 1,000 reviews in which we manually annotated 1,521 pairs of features and users' associated sentiment. We then used this dataset to evaluate the feature identification capabilities of all three approaches and the sentiment analysis capabilities of GuMa and ReUS.

The study indicates that feature extraction techniques are not yet effective enough to be used in practice [22,31] and that have lower precision and recall than reported in their initial studies. Our study also indicates that feature-specific sentiment analysis techniques have limited precision and recall, particularly for negative sentiments.

In the second study, we have evaluated three approaches searching for feature-related reviews: Lucene [23], a search engine library relying on vector space model and Okapi BM25 similarity; MARAM [14] adopting Jaccard Similarity; and SAFE, exploiting part-of-speech parsing and semantic similarity measure [15]. With human-coders' help, we elaborated a novel evaluation dataset, including 1,113 annotated reviews with respect to 24 queries (app features). We used the dataset to evaluate the approaches searching for feature-related reviews.

The findings showed Lucene, a standard searching tool, provides better performance than state-of-the-art techniques developed for app review analysis. We concluded the tool provides promising accuracy, and could be potentially used to support requirements elicitation use cases. We suggest future studies focus on extending existing techniques; and pay more attention to the efficiency of their approaches.

We hope our replication package [24] consisting of two novel annotated datasets, the evaluation methods, and our reimplementation of tools mining user opinions and searching for feature-related reviews will contribute to improving the quality of app review mining techniques.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We shared the link to our replication package (https://github.com/jsdabrowski/IS-22/); contains datasets and reimplementation of tools mining user opinions and searching for feature-related reviews.

References

[1] W. Martin, F. Sarro, Y. Jia, Y. Zhang, M. Harman, A survey of app store analysis for software engineering, IEEE Trans. Softw. Eng. 43 (9) (2017) 817–847, doi.ieeecomputersociety.org/10.1109/TSE.2016.2630689.

[2] L. Ceci, Number of apps available in leading app stores as of 2021, 2021, https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (Accessed: 20 Jan 2021).

[3] A. AlSubaihin, F. Sarro, S. Black, L. Capra, M. Harman, App store effects on software engineering practices, IEEE Trans. Softw. Eng. (2019) 1, http://dx.doi.org/10.1109/TSE.2019.2891715.

[4] J. Dąbrowski, E. Letier, A. Perini, A. Susi, Mining user feedback for software engineering: Use cases and reference architecture, in: 30th IEEE International Requirements Engineering Conference, RE 2022, Melbourne, Australia, August (2022) 15-19, IEEE, 2022, pp. 114–126, http://dx.doi.org/10.1109/RE54965.2022.00017.

[5] J. Dąbrowski, Mining App Reviews to Support Software Engineering (Ph.D. thesis), University College London, 2022, https://discovery.ucl.ac.uk/id/eprint/10149747/.

[6] D. Pagano, W. Maalej, User feedback in the appstore: An empirical study, in: 2013 21st IEEE International Requirements Engineering Conference, RE, 2013, pp. 125–134.

[7] J. Dąbrowski, E. Letier, A. Perini, A. Susi, Analysing app reviews for software engineering: A systematic literature review, Empir. Softw. Eng. 27 (2022) 43, http://dx.doi.org/10.1007/s10664-021-10065-7.

[8] J.O. Johanssen, A. Kleebaum, B. Bruegge, B. Paech, How do practitioners capture and utilize user feedback during continuous software engineering? in: 2019 IEEE 27th International Requirements Engineering Conference, 2019.

[9] A. Begel, T. Zimmermann, Analyze this! 145 questions for data scientists in software engineering, in: 36th International Conference on Software Engineering, 2014, pp. 12–13.

[10] R.P.L. Buse, T. Zimmermann, Information needs for software development analytics, in: 34th International Conference on Software Engineering, 2012, pp. 987–996, http://dx.doi.org/10.1109/ICSE.2012.6227122.

[11] E.C. Groen, N. Seyff, R. Ali, F. Dalpiaz, J. Doerr, E. Guzman, M. Hosseini, J. Marco, M. Oriol, A. Perini, M. Stade, The crowd in requirements engineering: The landscape and challenges, IEEE Softw. 34 (2) (2017) 44–52, http://dx.doi.org/10.1109/MS.2017.33.

[12] App Annie, 2020, https://www.appannie.com/ (Accessed: 20 Jan 2022).

[13] J. Dąbrowski, E. Letier, A. Perini, A. Susi, Finding and analyzing app reviews related to specific features: A research preview, in: 25th International Working Conference, REFSQ 2019, Essen, Germany, March (2019) 18-21, Proceedings, 2019, pp. 183–189, http://dx.doi.org/10.1007/978-3-030-15538-4_14.

[14] C. Iacob, S. Faily, R. Harrison, Maram: Tool support for mobile app review management, in: Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services, MobiCASE '16, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2016, pp. 42–50.

[15] T. Johann, C. Stanik, A.M.A.B., W. Maalej, Safe: A simple approach for feature extraction from app descriptions and app reviews, in: 2017 IEEE 25th International Requirements Engineering Conference, 2017, pp. 21–30, http://dx.doi.org/10.1109/RE.2017.71.

[16] B. Liu, Sentiment analysis and opinion mining, in: Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers, 2012, http://dx.doi.org/10.2200/S00416ED1V01Y201204HLT016.

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

[18] M. Dragoni, M. Federici, A. Rexha, An unsupervised aspect extraction strategy for monitoring real-time reviews stream, Inf. Process. Manage. 56 (3) (2019) 1103–1118, http://dx.doi.org/10.1016/j.ipm.2018.04.010.

[19] J. Dąbrowski, E. Letier, A. Perini, A. Susi, Mining user opinions to support requirement engineering: An empirical study, in: S. Dustdar, E. Yu, C. Salinesi, D. Rieu, V. Pant (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2020, pp. 401–416.

[20] X. Gu, S. Kim, What parts of your apps are loved by users? (t), in: 30th International Conference on Automated Software Engineering, 2015, pp. 760–770, http://dx.doi.org/10.1109/ASE.2015.57.

[21] F. Shah, K. Sirts, D. Pfahl, Simulating the impact of annotation guidelines and annotated data on extracting app features from app reviews, in: Proceedings of the 14th International Conference on Software Technologies, ICSOFT 2019, SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 2019, pp. 384–396, http://dx.doi.org/10.5220/0007909703840396.

[22] F.A. Shah, K. Sirts, D. Pfahl, [Is the safe] approach too simple for app feature extraction? A replication study, in: 25th International Working Conference, REFSQ 2019, Essen, Germany, March (2019) 18-21, Proceedings, 2019, pp. 21–36, http://dx.doi.org/10.1007/978-3-030-15538-4_2.
[23] M. McCandless, E. Hatcher, O. Gospodnetic, Lucene in Action, Second Edition: Covers Apache Lucene 3.0, Manning Publications Co., USA, 2010.
[24] J. Dąrowski, Replication package for a journal paper in information systems - mining and searching app reviews for requirements engineering, 2022, https://github.com/jsdabrowski/IS-22/.
[25] D.S. Batory, Feature models, grammars, and propositional formulas, in: J.H. Obbink, K. Pohl (Eds.), Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September (2005) 26-29, Proceedings, in: 3714 of Lecture Notes in Computer Science, Springer, 2005, pp. 7–20, http://dx.doi.org/10.1007/11554844_3.
[26] K.E. Wiegers, J. Beatty, Software Requirements 3, Microsoft Press, USA, 2013.
[27] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. Rep. CMU/SEI-90-TR-021,, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.
[28] E.C. Groen, S. Kopczyńska, M.P. Hauer, T.D. Krafft, J. Doerr, Users – the hidden software product quality experts?: A study on how app users report quality aspects in online reviews, in: 2017 IEEE 25th International Requirements Engineering Conference, RE, 2017, pp. 80–89, http://dx.doi.org/10.1109/RE.2017.73.
[29] N. Jha, A. Mahmoud, Mining non-functional requirements from app store reviews, Empir. Softw. Eng. 24 (6) (2019) 3659–3695.
[30] S. Lim, A. Henriksson, J. Zdravkovic, Data-driven requirements elicitation: A systematic literature review, SN Comput. Sci. 2 (1) (2021) 16.
[31] F. Dalpiaz, M. Parente, [Re-SWOT:] From user feedback to requirements via competitor analysis, in: 25th International Working Conference, REFSQ 2019, Essen, Germany, March (2019) 18-21, Proceedings, 2019, pp. 55–70, http://dx.doi.org/10.1007/978-3-030-15538-4_4.
[32] C.D. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, MIT Press, Cambridge, MA, USA, 1999.
[33] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, A. Kappas, Sentiment strength detection in short informal text, J. Am. Soc. Inf. Sci. Technol. 61 (12) (2010) 2544–2558, https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.21416.
[34] NLTK: Sample usage for collocations, 2022, https://www.nltk.org/howto/collocations.html.
[35] Apache lucene 7.4.0 documentation, 2022, https://lucene.apache.org/core/7_4_0/index.html.
[36] P.M. Vu, T.T. Nguyen, H.V. Pham, T.T. Nguyen, Mining user opinions in mobile app reviews: A keyword-based approach (t), in: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE, ASE '15, 2015, pp. 749–759, http://dx.doi.org/10.1109/ASE.2015.85.
[37] J. McAuley, C. Targett, Q. Shi, A. van den Hengel, Image-based recommendations on styles and substitutes, in: 38th International Conference on Research and Development in Information Retrieval, ACM, 2015, pp. 43–52, http://dx.doi.org/10.1145/2766462.2767755.
[38] J. Pustejovsky, A. Stubbs, Natural Language Annotation for Machine Learning - a Guide to Corpus-Building for Applications, O'Reilly, 2012.
[39] H. Cunningham, D. Maynard, V. Tablan, C. Ursu, K. Bontcheva, Developing Language Processing Components with GATE Version 8, University of Sheffield Department of Computer Science, 2014.
[40] B. Croft, D. Metzler, T. Strohman, Search Engines: Information Retrieval in Practice, first ed., Addison-Wesley Publishing Company, USA, 2009.
[41] L. Hoon, R. Vasa, J.-G. Schneider, K. Mouzakis, A preliminary analysis of vocabulary in mobile app user reviews, in: Proceedings of the 24th Australian Computer-Human Interaction Conference, OzCHI '12, ACM, New York, NY, USA, 2012, pp. 245–248, http://dx.doi.org/10.1145/2414536.2414578.
[42] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, R. Oliveto, Sentiment analysis for software engineering: How far can we go? in: 40th International Conference on Software Engineering, 2018, pp. 94–104, http://dx.doi.org/10.1145/3180155.3180195.
[43] F. Calefato, F. Lanubile, F. Maiorano, N. Novielli, Sentiment polarity detection for software development, Empirical Softw. Engg. 23 (3) (2018) 1352–1382, http://dx.doi.org/10.1007/s10664-017-9546-9.
[44] D.M. Berry, J. Cleland-Huang, A. Ferrari, W. Maalej, J. Mylopoulos, D. Zowghi, Panel: Context-dependent evaluation of tools for nl re tasks: Recall vs. precision, and beyond, in: 2017 IEEE 25th International Requirements Engineering Conference, RE, 2017, pp. 570–573, http://dx.doi.org/10.1109/RE.2017.64.
[45] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer Publishing, Company, Incorporated, 2012.
[46] A.A. Al-Subaihin, Software Engineering in the Age of App Stores: Feature-Based Analyses to Guide Mobile Software Engineers, (Ph.D. thesis, doctoral thesis), University College London, 2019.
[47] C.D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, USA, 2008.
[48] R.R. Korfhage, Information Storage and Retrieval, John Wiley & Sons, 1997.
[49] P. Achimugu, A. Selamat, R. Ibrahim, M.N. Mahrin, A systematic literature review of software requirements prioritization research, Inf. Softw. Technol. 56 (6) (2014) 568–585, http://dx.doi.org/10.1016/j.infsof.2014.02.001.
[50] W. Martin, M. Harman, Y. Jia, F. Sarro, Y. Zhang, The app sampling problem for app store mining, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 123–133, http://dx.doi.org/10.1109/MSR.2015.19.