How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews

Article · September 2014

DOI: 10.1109/RE.2014.6912257

How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews

Emitza Guzman
Technische Universität München
Garching, Germany
emitza.guzman@mytum.de

Walid Maalej
University of Hamburg
Hamburg, Germany
maalej@informatik.uni-hamburg.de

***Abstract*—App stores allow users to submit feedback for downloaded apps in form of star ratings and text reviews. Recent studies analyzed this feedback and found that it includes information useful for app developers, such as user requirements, ideas for improvements, user sentiments about specific features, and descriptions of experiences with these features. However, for many apps, the amount of reviews is too large to be processed manually and their quality varies largely. The star ratings are given to the whole app and developers do not have a mean to analyze the feedback for the single features. In this paper we propose an automated approach that helps developers filter, aggregate, and analyze user reviews. We use natural language processing techniques to identify fine-grained app features in the reviews. We then extract the user sentiments about the identified features and give them a general score across all reviews. Finally, we use topic modeling techniques to group fine-grained features into more meaningful high-level features. We evaluated our approach with 7 apps from the Apple App Store and Google Play Store and compared its results with a manually, peer-conducted analysis of the reviews. On average, our approach has a precision of 0.59 and a recall of 0.51. The extracted features were coherent and relevant to requirements evolution tasks. Our approach can help app developers to systematically analyze user opinions about single features and filter irrelevant reviews.**

I. Introduction

Application distribution platforms, or app stores, allow users to search, buy, and deploy software apps for mobile devices with a few clicks. These platforms also allow users to share their opinion about the app in text reviews, where they can, e.g., express their satisfaction with a specific app feature or request a new feature. Recent empirical studies [4], [6], [21] showed that app store reviews include information that is useful to analysts and app designers, such as user requirements, bug reports, feature requests, and documentation of user experiences with specific app features. This feedback can represent a "voice of the users" and be used to drive the development effort and improve forthcoming releases [15], [25].

However, there are several limitations which prevent analysts and development teams from using the information in the reviews. First, app stores include a *large amount* of reviews, which require a large effort to be analyzed. A recent study found that iOS users submit on average 22 reviews per day per app [21]. Very popular apps such as Facebook get more than 4000 reviews per day. Second, the *quality* of the reviews varies widely, from helpful advice and innovative ideas to insulting comments. Third, a review typically contains a *sentiment mix* concerning the different app features, making it difficult to, e.g., filter positive and negative feedback or retrieve the feedback for specific features. The usefulness of the star ratings in the reviews is limited for development teams, since a rating represents an average for the whole app and can combine both positive and negative evaluations of the single features.

To reduce the effort spent in collecting and understanding user feedback from the app reviews, we propose an approach that automatically extracts app features referred in the reviews together with the user opinions about them. We refer to a feature as a prominent or distinctive visible characteristic or quality of an app [10]. It can be any description of specific app functionality visible to the user (e.g., “uploading files” or “sending a message”), a specific screen of the app (e.g., “configuration screen”), a general quality of the app (e.g., “load time”, “size of storage”, or “price”), as well as specific technical characteristics (e.g., “encryption technology”).

Our approach produces a fine-grained list of features mentioned in the reviews. It then extracts the user sentiments of the identified features and gives them a general score across all reviews. Finally, it groups fine-grained features into more meaningful high-level features that tend to be mentioned in the same reviews and shows the opinions of users about these high-level features. We use collocation finding [17] for extracting the fine-grained features, sentiment analysis [28] for extracting the sentiments and opinions associated to the features, and topic modeling [2] for the grouping of related features.

We evaluated the approach with 32210 reviews for seven iOS and Android apps and compared the results with 2800 manually peer-analyzed reviews. The results show that our approach successfully extracts the most frequently mentioned features, that the groups of features are coherent and relevant to app requirements, and that the sentiment analysis results positively correlate to the manually assigned sentiment scores.

The contribution of this paper is threefold. First, we introduce an approach to automatically extract fine-grained and course-grained features from the text of app reviews. Second, we present a method for aggregating the sentiments of many users for a feature by applying automated sentiment analysis techniques on app reviews. Finally, we describe a detailed evaluation method based on content analysis, a manually created evaluation dataset, and an evaluation tool, which can all be used for similar evaluations.

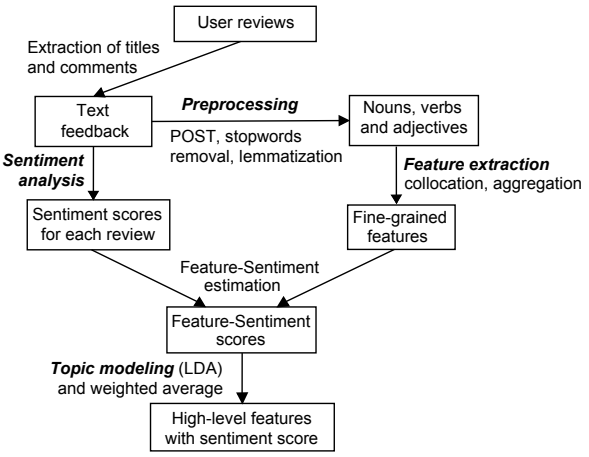

Fig. 1: Overview of the approach.

The remainder of the paper is structured as follows. Section II introduces the approach. Section III describes the evaluation method and the content analysis study. Section IV reports on the results. Section V discusses the findings, their implications, and limitations. Finally, Section VI summarizes the related work, while Section VII concludes the paper.

II. Approach

The main goal of our approach is to automatically identify application features mentioned in user reviews, as well as the sentiments and opinions associated to these features. For this we use Natural Language Processing, and Data Mining techniques. Figure 1 shows an overview of the approach. First, we collect the user reviews for a specific app and extract the title and text comments from each review. Then, we *preprocess* the text data to remove the noise for feature extraction. Afterwards, we extract the features from the reviews by applying a *collocation finding* algorithm and aggregating features by their meaning. This produces the list of fine-grained features, each consisting of two keywords. To extract the sentiments about the feature we apply *lexical sentiment analysis* to the raw data from the titles and comments. Lexical sentiment analysis assigns a sentiment score to each sentence in the review. When a feature is present in the sentence, the sentiment score of the sentence is assigned to the feature. Finally, we apply a *topic modeling* algorithm to the extracted features and their associated sentiment scores to create a more coarse-grained summary, which groups features that are mentioned in the same reviews. In the following we explain the main steps of our approach.

A. Data Collection and Preprocessing

In the first step we collect and preprocess the user reviews. When developing and evaluating our approach we used reviews from the Apple App Store[1] and Google Play[2]. However, the approach can also be applied to reviews from other platforms. For collecting the Apple Store data we use a modified version of an open source scraping tool[3]. For the collection of Google Play reviews we developed a tool which uses the Google Play Application Programming Interface (API). We store the collected data in a MySQL database. After gathering the data we extract the title and comments from each review as both might include references to features or sentiments.

While the sentiment analysis does not require further preprocessing, the feature extraction requires three additional steps:

- **Noun, verb, and adjective extraction.** We use the part of speech (POS) tagging functionality of the Natural Language Toolkit, NLTK[4], for identifying and extracting the nouns, verbs, and adjectives in the reviews. We assume that these parts of speech are the most likely to describe features as opposed to others such as adverbs, numbers, or quantifiers. A manual inspection of 100 reviews confirmed this assumption.
- **Stopword removal.** We remove stopwords to eliminate terms that are very common in the English language (e.g., “and”, “this”, and “is”’). We use the standard list of stopwords provided by Lucene[5] and expand it to include words that are common in user reviews, but are not used to describe features. The words we added to the stopword list are the name of the application itself, as well as the terms "app", "please", and "fix".
- **Lemmatization.** We use the Wordnet [19] lemmatizer from NLTK for grouping the different inflected forms of words with the same part of speech tag which are syntactically different but semantically equal. This step reduces the number of feature descriptors that need to be inspected later. With this process, for example, the terms describing the verbs "sees" and "saw" are grouped into the term "see".

B. Feature Extraction

We use the collocation finding algorithm provided by the NLTK toolkit for extracting features from the user reviews. A collocation is a collection of words that co-occur unusually often [1]. Manning and Schütze define collocations as expressions of two or more words which correspond to a conventional way of referring to things [17]. An example of a collocation is <*strong tea*>, whereas <*powerful tea*> is not a collocation since these two words are not normally used in the English language together. Collocations do not necessarily imply that the words are adjacent. Several words can be between the words that constitute the collocation e.g., <*knock door*>. Features can generally be described as collocations, as they are normally a collection of terms that are used repeatedly to convey a specific meaning. Examples of collocations that identify application features are <*pdf viewer*>, <*user interface*> and <*view picture*>. We use a likelihood-ratio test [17] for finding collocations consisting of two words in our reviews.

[1]https://itunes.apple.com/us/genre/ios/id36
[2]https://play.google.com/store?hl=en
[3]https://github.com/oklahomaok/AppStoreReview
[4]http://nltk.org/
[5]https://lucene.apache.org/

TABLE I: Examples of SentiStrength scores in the user reviews.

Sentence in review	Word scores	Sentence score
had fun using it before but now its really horrible :(help!!	had fun[2] using it before but now its really horrible[-4] [-1 booster word] :[[-1 emoticon] help!![-1 punctuation emphasis]	{2,-5}
uploading pictures with the app is so annoying!	uploading pictures with the app is so annoying[-3]! [-1 punctuation emphasis]	{1,-3}
pleeeeease add an unlike button and I will love you forever!!	pleeeeease[3] [+0.6 spelling emphasis] add an unlike button and I will love[3] you forever!![+1 punctuation emphasis]	{5, -1}

After finding the collocations we filter them by taking into consideration only those that appear in at least three reviews and that have less than three words (nouns, verbs, or adjectives) distance between them. Typically, the order of the words is important for collocations. However, we consider word ordering unimportant for describing features. For example, we consider that the pairs of words <*pdf viewer*> and <*viewer pdf*> convey the same meaning, although the latter might be used less frequently. Therefore, we merge collocations that consist of the same words but have different ordering.

Users can use different words to refer to the same features. We group collocations whose pairs of words are synonyms and use Wordnet as a synonym dictionary. Wordnet also allows us to group collocations with misspellings together.

When grouping features together we consider the word collection with the highest frequency to be the name of the feature. For example, if we have the following word collections: <*picture view*>, <*view photographs*> and <*see photo*> with a frequency of 30, 10, and 4 respectively. Our approach would then group these features together since they are synonyms. Afterwards, it would choose the one with the highest frequency as the name for the feature, in this case <*picture view*>.

C. Sentiment Analysis

Sentiment analysis is the process of assigning a quantitative value (positive or negative) to a piece of text expressing an affect or mood [12]. For analyzing sentiments in user reviews, we use SentiStrength [28], a lexical sentiment extraction tool specialized in dealing with short, low quality text. Previous research has shown that SentiStrength has a good accuracy for short texts in social media, such as, Twitter and movie reviews [27]. Pagano and Maalej [21] found that 80.4% of the comment reviews in the App Store contain less than 160 characters, making SentiStrength a good candidate for analyzing sentiments in user reviews.

SentiStrength divides the review text into sentences and then assigns a positive and negative value to each sentence. The idea behind this is that humans can express both positive and negative sentiments in the same sentence, e.g., "I loved the app until this last release, it's awful now". SentiStrength assigns positive scores in the $[+1, +5]$ range, where $+5$ denotes an extremely positive sentiment and 1 denotes the absence of sentiment. Similarly, negative sentiments range from $[-1, -5]$, where -5 denotes an extremely negative sentiment and -1 indicates the absence of any negative sentiment.

Furthermore, SentiStrength assigns fixed scores to tokens in a dictionary where common emoticons are also included. For example, "love" is assigned a score of $\{3, -1\}$ and "hate" a $\{1, -4\}$ score. Only words that are present in the dictionary are attributed with an individual score. Modifier words and symbols also alter the score. For example, "absolutely love" is assigned a score of $\{4, -1\}$. The same score is given to "looove" and "love!!!". Additionally, it offers the possibility to extend its dictionary. However, for our approach we used the dictionary provided by the tool. The sentiment score of the whole sentence is computed by taking the maximum and minimum scores among all the words in a sentence. Table I shows three examples of sentences scored by SentiStrength.

After calculating the sentiment score in the sentences, we compute the sentiment score for the features. We consider the sentiment score of a feature to be equal to the positive or negative score of the sentence in which it is present. As a feature score we choose the score with the maximum absolute value. In the case that the positive and negative values are the same, we assign the negative value to the feature. For example, lets consider the sentence "Uploading pictures with the app is so annoying!". This sentence contains the feature <*uploading pictures*>, and the sentiment score of the sentence is {1,-3}. Therefore, we assign the feature the sentiment score of -3. This step produces a list of all extracted features, their frequencies (how often they were mentioned), and their sentiment scores. This list is a fine-grained summary which developers and requirement analysts can use to get a detailed overview of users' primary concerns.

D. Topic Modeling

Our approach produces a high-level summary as a final result. This summary contains groups of different features and a corresponding sentiment score. To group features that tend to co-occur in the same reviews we use Latent Dirichlet Allocation (LDA) [2], a topic modeling algorithm. LDA is a probabilistic distribution algorithm which uses Gibbs sampling to assign topics to documents, in our case user reviews. In LDA a topic is a probabilistic distribution over words and each document is modeled as a mixture of topics. This means that each review can be associated to different topics and that topics are associated to different words with a certain probability. An example of a topic can be the set of words *{crash, update, frustrated, newest, version, help, bug}* which describes users' experiences when updating a faulty app.

We used the Matlab Topic Modeling Toolbox[6]. Instead of inputting the words forming the vocabulary of our analyzed reviews to the LDA algorithm, as is normally the case when applying LDA, we input the list of extracted features and model each feature as a single word. For example, the feature described with the <*picture view*> collocation is transformed into the single term $picture\_view$. LDA then outputs the feature distribution of each topic and the probabilistic topic composition for each review. An example of a topic with this modification could then be the set of features *{picture_view, camera_picture, upload_picture, delete_picture}* which describes features related to manipulating pictures in an application.

We calculate topic sentiments as follows. Let $R = \{r_1, r_2, ..., r_n\}$ be the set of analyzed reviews and $T = \{t_1, t_2, ..., t_m\}$ the set of extracted topics. The final output of the LDA computation is the matrix $W_{n \times m}$, where $w_{i,j}$ contains the number of times a feature mentioned in review r_i is associated with topic t_j. We then use a weighted average to calculate the sentiment score of each topic. For every topic t_j we calculate the topic sentiment score ts_j as:

$$ts_j = \frac{\sum_{i=1}^{n} w_{i,j} \cdot s_i}{\sum_{i=1}^{n} w_{i,j}}$$

where $S = \{s_1, s_2, ..., s_l\}$ denotes the sentiment score of each feature associated to the topic t_j.

III. Evaluation Method

Our evaluation goal was twofold. We aimed at evaluating (a) the relevance of the automatically identified features from the reviews and (b) the correctness of the automatically calculated sentiment estimation for each feature over all reviews. The specific evaluation questions were:

1) Does the extracted text represent real app features?
2) Are the extracted and grouped features coherent and relevant for app analysts and developers?
3) Is the automated sentiment estimation similar to a manually conducted sentiment assessment?

To answer these questions, we created a truth set through a careful, manual, peer-conducted content analysis process [20]. We then compared the results of our approach against the manual analysis. In the following we describe the dataset that was used in our evaluation, how the truth set was created, as well as the quality metrics that we considered for the evaluation. To encourage replication, we make the evaluation data and evaluation tools publicly available on the project website[7].

A. Dataset

Our evaluation data consisted of user reviews from the US App Store and Google Play. The App Store is an application distribution platform for Apple devices, whereas Google Play distributes applications for Android devices. Both app stores allow users to write reviews about the downloaded apps. Additionally, both stores cluster the apps into different categories based on functionality. For our evaluation, we selected seven apps from the lists of "most popular apps" in different categories. Table II shows these apps, their categories, and the number of reviews considered in the evaluation.

TABLE II: Overview of the evaluation apps.

App	Category	Platform	#Reviews	∅Length
AngryBirds	Games	App Store	1538	132
Dropbox	Productivity	AppStore	2009	172
Evernote	Productivity	App Store	8878	200
TripAdvisor	Travel	App Store	3165	142
PicsArt	Photography	Google Play	4438	51
Pinterest	Social	Google Play	4486	81
Whatsapp	Communication	Google Play	7696	38

For the App Store we selected the apps AngryBirds, Dropbox, Evernote, and TripAdvisor from the categories Games, Productivity, and Travel. We collected all user reviews written in 2013 for these apps. For Google Play we selected the apps PicsArt, Pinterest, and Whatsapp from the categories Photography, Social, and Communication. We collected the maximum number of reviews allowed by the Google Play APIs. These were the most recent 4000 to 7000 reviews for the apps, as of February 2014.

We selected different categories of apps because we wanted to evaluate our approach against reviews containing diverse vocabularies, describing different features, and written by different user audiences. Users from Angrybirds, Dropbox, and TripAdvisor might have different expectations, interact with technology in various manners, belong to different age groups, and express their sentiments and experiences in different ways.

We chose popular apps to increase the probability that the people creating the truth set were familiar with the apps, reducing the manual feature extraction effort and minimizing errors during the truth set creation. Popular apps are also more likely to have more reviews. An automated analysis for these apps would probably be more realistic and useful. On average, the reviews of the App Store apps had 178 characters, while Google Play apps included only 53 characters.

For each user review we collected the title, comment, date, author, version, and star rating. We ran our approach on the text in the title and comment of each review. To compare the results generated by our approach with human assessments, we created a truth set of the features mentioned in the review and their associated sentiments.

B. Truth Set Creation

The methodological cornerstone for the creation of the truth set was the use of content analysis techniques as described by Neuendorf [20] and by Maalej and Robillard [16]. This process involved the systematic assessment of a reviews sample by

[6]http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm
[7]http://mobis.informatik.uni-hamburg.de/app-review-analysis/

human coders, who read each review and assessed its contents according to a strict coding guide. Apart from both authors, this process involved seven trained coders who independently coded 2800 randomly sampled user reviews totaling 60,738 words. For each user review, two coders independently:

1) indicated whether the review contained a feature request or feedback about an existing feature,
2) identified the app features mentioned in the review, and
3) assessed the sentiments associated to each feature.

The first step of the truth set creation consisted of developing the coding guide. We needed the guide because the review content and the coding task can be interpreted differently by distinct coders. The goal of a coding guide is to systemize the task and minimize disagreements between peer-coders. The guide contained instructions about the task, clear definitions of a feature, feature request, feedback on a feature, and of the different sentiment scales. The guide also included examples for each possible assessment and rules to follow. The coding guide was created in an iterative process including four main iterations. In each iteration, we tested the guide by independently peer-assessing 50 reviews.[8] The coders' disagreement was manually analyzed and the coding guide was modified (e.g., include more examples and improve the definitions) to avoid similar disagreements in the next iterations.

The second step consisted of the sampling. We selected 400 reviews for each of the evaluation apps based on stratified random sampling [23]. Our sampling scheme took into account the rating distribution of the specific apps. For instance, assume that the app has 1000 reviews, the sample size is of 100, and the rating distribution in all reviews is, e.g., as follows: 40% of the reviews had 5 stars, 40% had 4 stars, 15% had 3 stars, 5% had 2 starts and 1% had one star. Then, the sample with 100 reviews includes 40 reviews of 5 stars, 40 of 4 stars, 15 of 3 stars, etc. The reviews in each strata were selected randomly from the corresponding group of ratings.

The third step consisted of peer-coding the reviews in the samples. For this task, we developed a coding tool, which displays a single review (title, comment, and rating) at a time together with the coding questions. Figure 2 shows a screenshot of the tool. When using the coding tool the coders could select the features from the review text by clicking on the words describing the feature. Additionally, the coders assigned a sentiment to each of the features. The sentiment scale was very positive, positive, neutral, negative, and very negative. Furthermore, the coders were asked to indicate if the review included feedback about an already existing feature, a request for a new feature, a bug report, or other type of content. Multiple selections were allowed. Coders were able to stop and resume their coding tasks at any time they wished.

The coders were graduate students from the Technical University of Munich and the University of Hamburg. They all had a high command of English and had software development experience. The reviews were randomly assigned to the coders, so that each review was assigned twice and that each coder shared a similar number of assignments with all other coders.

Each coder received the coding guide, the tool, and the coding assignments. Moreover, we explained the coding task in a short meeting and we were available for clarification requests during the coding period. To assure that the coders had some knowledge about the evaluation apps, we asked them to read the app store descriptions of each of the seven apps.

We also asked the coders to record the total time spent on their coding assignments in order to estimate the amount of effort necessary for a fine-grained manual analysis of user reviews. Coders reported spending between 8 and 12.5 hours for coding about 900 reviews. These numbers confirm previous studies, which have highlighted the large amount of effort needed to manually analyze user feedback [4], [21].

The final step in the truth set creation consisted of the analysis of disagreements. Overall 30% of the coded reviews included feedback about features, while 10% included feature requests. The average sentiment for all coded features was 0.17 (i.e., neutral). These results confirm the findings of previous exploratory studies [21]. The coders identified a total of 3005 features, agreeing on 1395 features (53%) and disagreeing on 1610 features (47%). The coders also agreed that 1086 reviews did not contain any features. This disagreement was handled in two steps. First, we randomly selected 100 reviews with disagreements and analyzed reasons, and common patterns for falsely classified features. We then used the results to automatically filter misclassified features. Second, one of the authors manually reviewed the remaining features with disagreement and decided if each of the previously labeled features was mentioned in the review or not. At the end, the truth set included 2928 features. We solved the disagreement between the sentiments associated to features by transforming the categorical values into numerical values and calculating the average of both coders.

C. Evaluation Metrics

We evaluate our topics using precision, recall, and F-measure. Precision is computed by dividing the number of true positives by the sum of true positives and false positives. Recall is computed by dividing the number of true positives by the sum of true positives and false negatives. We use the general form of the F-measure, which combines the precision and recall results, for its computation.

We define a feature as true positive, if it was automatically extracted from a review and was also manually identified in that review. False positives are features that were automatically associated to a review in one of the topics, but were not identified manually in that review. Finally, false negative features were manually identified in a review but were not present in any of the extracted topics associated to the review.

In addition to precision, recall, and F-measure we used two further metrics for determining the quality of the topics (groups of features) generated by the LDA algorithm. The *coherence* of topics assesses how well the topics are logical and consistent and whether they share a common theme. The *requirements*

[8]The test reviews were different from the reviews in the evaluation samples.

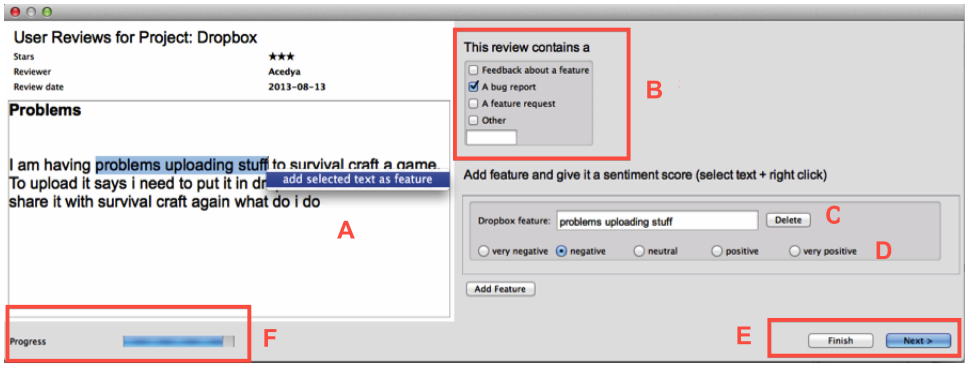

Fig. 2: Coding tool for the creation of the truth set (available on the project website). A: the review text to be coded, B: the type of the review, C: features references in the review, D: sentiment about the feature, E: navigation button, and F: progress bar.

relevance measures whether the topics contain information that help define, understand, and evolve the requirements of the app. These last two metrics were qualitatively evaluated by the authors with a 5-level scale.

IV. Evaluation Results

We first report on the feature extraction results and then on the evaluation of the sentiment estimations.

A. Feature Extraction

We evaluate the high-level topics generated by our approach and not the fine-grained list of features. The topics contain the fine-grained features, therefore, the results from the topic evaluation reflect the performance of the fine-grained feature extraction. Due to the large amount of fine-grained features (e.g., 3700 for Evernote) a manual qualitative evaluation would be unfeasible and the consideration of the N-top features would produce unwanted bias.

After running the feature extraction step of our approach we obtained a list of wordsets designating app features. Many of the wordsets extracted as features contained words that do not describe features but rather the opinions or sentiments of users (e.g., great, bad, good, like, hate...). To filter these words we decided to slightly modify our approach and include all words that are assigned a sentiment by the lexical sentiment analysis tool into the stopword list. We use F_S to refer to the original approach which includes words with a sentiment meaning into the feature extraction algorithm, and F_{NS} to refer to the modified approach which excludes sentiment words when generating feature descriptors. Examples of the most common features extracted by the F_{NS} approach for two of the applications are shown in Figure 3.

Table III shows the number of different features extracted for each app for the original approach F_S and for the modified approach F_{NS}. The number of extracted feature varies between 181 and 3700 features. To deal with this large amount of information, we gather the extracted features into high-level groups using topic modeling.

We calculated the precision, recall and F-measure of the extracted topic models for the seven evaluation apps. We generated 20 topic models for each app. Table V and Table VI show examples of topics and their associated sentiments for the Dropbox and Pinterest apps respectively. We compared the results when using the F_{NS} approach which excludes words with sentiments from the feature extraction, and F_S which includes them. Table IV summarizes the results. Both approaches had similar results, varying on a project basis. We achieved the highest precision of 91% for Whatsapp with F_{NS} and the highest recall of 73% for the same app with F_{NS}. On

TABLE III: Number of extracted fine-grained features per app.

App	F_S	F_{NS}
AngryBirds	284	219
Dropbox	612	600
Evernote	3700	3127
TripAdvisor	846	754
PicsArt	290	181
Pinterest	625	465
Whatsapp	383	234

TABLE IV: Precision, Recall, and F-measure for the high-level feature extraction with topic modeling.

App	Precision	Recall	F-measure
	F_S		
AngryBirds	0.335	0.332	0.334
Dropbox	0.608	0.475	0.533
Evernote	0.474	0.416	0.443
TripAdvisor	0.421	0.399	0.410
PicsArt	0.750	0.669	0.707
Pinterest	0.644	0.623	0.634
Whatsapp	0.843	0.728	0.781
F_S Average	**0.582**	**0.520**	**0.549**
	F_{NS}		
AngryBirds	0.368	0.321	0.343
Dropbox	0.603	0.473	0.531
Evernote	0.451	0.389	0.418
TripAdvisor	0.403	0.370	0.386
PicsArt	0.815	0.661	0.730
Pinterest	0.658	0.592	0.623
Whatsapp	0.910	0.734	0.813
F_{NS} Average	**0.601**	**0.506**	**0.549**

TABLE V: Most common topics extracted from the user reviews of the Dropbox app with their sentiments

Topic	Senti. score
upload_photo, load_photo, photo_take, photo_want, upload_want, download_photo, upload_feature, move_photo, keep_upload, keep_try	1.51 *Positive*
file_name, folder_file, rename_file, file_add, folder_rename, make_folder, file_change, change_name, file_copy, option_folder	1.49 *Positive*
file_load, video_load, video_upload, download_file, download_video, download_phone, download_time, file_phone, update_need, computer_phone	1.75 *Positive*

TABLE VI: Most common topics extracted from the user reviews of Pinterest with their sentiments

Topic	Senti. score
board_pin, pin_wish, make_board, create_board, sub_board, create_pin, use_board, edit_board, button_pin, edit_pin	2.47 *Very Positive*
pin_see, pin_go, load_pin, keep_thing, board_change, pin_scroll, click_pin, pin_everything, time_search, browse_pin	2.28 *Very Positive*
craft_idea, craft_recipe, home_idea, idea_diy, look_idea, everything_want, home_decor, thing_internet, art_craft, load_image	2.23 *Very Positive*

average, the precision of F_{NS} was approximately 60% and the recall was about 50%. The average precision of F_S was 58% and its recall was of 52%. We observed the lowest precision and recall for AngryBirds, an iOS game, which also resulted in the largest amount of disagreement during the truth set creation. Android Apps had a higher precision and recall than iOS apps.

For measuring the coherence and the requirements relevance, we manually examined the 20 topics generated by the F_{NS} version of our approach for each evaluation app. We also examined the 10 features which were most strongly associated to each of the topics. Table VII summarizes the results.

Coherence: To qualitatively measure the coherence of the topics generated by our approach we manually analyzed the ten most popular features for each topic. We evaluated the coherence of each topic by analyzing if the features conforming the topic shared a common theme. Then, we rated the coherence of each topic on a 5-level scale (from *very good* to *very bad*). Afterwards, we converted the individual ratings for each topic into a numerical scale ([-2,2] range) and calculated a coherence average for each app. The approach had a *good* to *neutral* coherence for all apps with the exception of Whatsapp. The difference in the coherence levels for Whatsapp can be explained by the average length of its reviews. These were much shorter than for the reviews of the other apps and LDA has a difficulty for generating meaningful topics on sparse data [26]. With the exception of Whatsapp, mixed topics (topics with no common theme) were not prevalent in the other apps.

TABLE VII: Coherence and requirements engineering relevance of topics.

App	Coherence	Req. Relevance
AngryBirds	Good	Good
Dropbox	Good	Very Good
Evernote	Good	Good
TripAdvisor	Good	Very Good
PicsArt	Neutral	Good
Pinterest	Good	Good
Whatsapp	Bad	Good

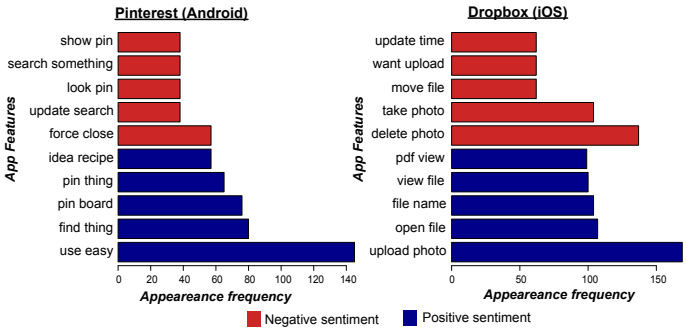

Fig. 3: Extracted features from Pinterest and Dropbox apps. Positive features are represented in blue, negative in red.

However, the presence of duplicate topics (topics sharing a very similar theme) was more prevalent in apps with less functionality or shorter length reviews, such as PicsArt and Whatsapp. LDA allows for the configuration of the number of topics and more experimentation with this variable could lead to less duplicate topics and different coherence results.

Requirements relevance: We evaluated the relevance of the extracted topics to the app requirements by manually analyzing the 10 most popular extracted features for each topic. Similarly to the coherence evaluation, we used a 5-level scale to rate whether a topic was relevant for requirements engineering. We considered a topic to be relevant to requirements when it mostly consisted of app features, app qualities, or information indicating how the users utilized the app.

Our approach generated topics with a *very good* to *good* relevance to requirements engineering for all apps. Even when the topics were not coherent, the 10 most popular extracted features were usually words describing actual app features and were important for understanding how users use the app. Each app usually had one or two topics with words that describe bug reports. These wordsets were usually not considered as features in our manual coding. However, they might contain valuable information for the evolution and maintenance of the app. The presence of noise (false positives) in the 10 most popular features (topics) was virtually non existent for all apps.

B. Sentiment Analysis

The approach proposed in this work assigns a feature the sentiment score of the sentence in which it is located. Figure 3

shows examples of extracted features and their associated sentiment. This information can be used to detect the features with the highest and lowest user approval. To compare the feature sentiment scores extracted by our approach with the ones given by the coders we converted the automatically extracted scores to categorical values. Similar to Kucuktunc et al. [12] we consider all reviews in the (2,5] range to be *very positive*, those in the (1,2] range *positive*, whereas those in the [-1,1] range are *neutral*. Reviews with a sentiment score in the [-2,-1) range are considered *negative* and those with a [-5,-2) range *very negative*. In this way we converted all sentiments computed by our approach into the scale used by the coders.

We then converted the categorical values into numerical values in the [-2,2] range, where -2 denotes a very negative sentiment and 2 a very positive sentiment. After that, we calculated the average sentiment score given by the two coders and used this as the "true sentiment" associated to each feature.

The Spearman's rho correlation coefficient between the sentence-based sentiment score and the truth set was of 0.445 (p-value< 2.2e-16), indicating a moderate positive correlation between them. Figure 3 shows examples of the most common positive and negative features for Dropbox and Pinterest.

We calculated an additional sentiment score for each feature by assigning each feature the sentiment score of the whole review in which it is mentioned. We compute the sentiment of an entire review by calculating the positive and negative average scores of all sentences in the review separately. For the case where both positive and negative sentence averages are in the [-1,1] range we assign the whole review the neutral score of 0. When the sentence negative average multiplied by 1.5 is less than the positive average, we assign the review the sentiment score of the negative average. In the opposite case, the review is assigned the positive average score. When comparing the positive and negative sentence average we multiply by 1.5 because negativity is considered to be less frequent in human written texts.[9]

With this review-based sentiment estimation, the positive correlation between the sentiment score and the truth set sentiment score was of 0.592 (p-value< 2.2e-16), representing a strong positive correlation.

One possible explanation for the higher correlation of the review-based sentiment score in comparison with the sentence-based approach is that people frequently use more than one sentence to express their opinion about a specific feature. That is, context is important when determining the sentiment associated to a specific feature.

V. Discussion

In this section we discuss our results and describe the limitations, threats to validity, and implications of our approach.

A. Result Interpretation

The qualitative and quantitative results of our approach are promising. Our approach was able to detect app features mentioned in the user reviews for a diverse set of apps belonging to different categories and serving different types of users. The feature extraction approach had a *good* performance for apps belonging to all analyzed categories with exception of the games category. One possible explanation can be the various ways users describe the gaming app features. Human coders had the largest difficulties when identifying features from this type of apps, as noted by their level of disagreement, confirming the challenges of examining this type of reviews.

The qualitative evaluation showed that the topics were coherent, contained little noise, and were relevant for requirements engineering tasks. Even for the apps where the recall was relatively low (i.e., 33-47%) the qualitative results showed that the topics accurately describe the overall functionality of the apps. Due to the inner workings of the LDA algorithm [2] the approach generates more coherent topics when dealing with lengthier reviews. Additionally, duplicate topics are more common for apps with less functionality and shorter reviews, such as PicsArts or Whatsapp. An advantage of our approach is that the number of topics (a parameter for LDA) can be manually tuned for each app by the project team to get less duplicate topics and better coherence and precision.

Finally, precision results were higher for apps with short reviews containing few or no features. One important finding is that our approach has a high performance for detecting reviews with no features mentioned in short reviews. This makes the approach useful for filtering non-informative reviews, which, e.g., only include praise or dispraise. These types of reviews tend to be very frequent in app stores [21]. Filtering them will help developers focus on the relevant and informative reviews.

B. Limitations

A limitation of our approach is that non frequently mentioned features are often not detected, as reflected by the recall values. This can be improved by including linguistic patterns, which describe the language structure in which features are described. This would allow for the identification of non common features through such patters in addition to the term frequency.

The used lexical sentiment analysis approach has the disadvantage of a limited handling of negation and conditionals, and no handling of past tense and sarcasm. However, the positive correlations found in both sentiment score approaches suggest that the noise produced by our approach due to this limitation is minimal. The expansion of the dictionary to include jargon common in user reviews, such as "bug", "crash", or "please fix!" could enhance the sentiment analysis performance. The outperformance of the review-based sentiment approach over the sentence-based approach confirms the importance of taking context into account when assigning sentiment scores to features.

C. Threats to Validity

The qualitative evaluation of the topic relevance to requirements engineering was done by the authors of the paper and not by actual developers of the apps. This is a threat to validity as the evaluators could be biased or could have incomplete

[9]See SentiStrength user manual http://sentistrength.wlv.ac.uk/

knowledge or misunderstandings about the specific information that developers working on these apps need with respect to requirements engineering. Another threat to validity is the high level of disagreement between (47%) coders of what constitutes an app feature. We tried to alleviate disagreement by providing a coding guide with a precise definitions and examples. Furthermore, we identified patterns for common human errors during the coding task. These human errors were a common cause for disagreement.

D. Implications

Our approach can be used to obtain two-level grained summaries of features mentioned in user reviews where the associated sentiment score gives an overall idea about the user's acceptance and evaluation of the features. A usage scenario could be a tool that visualizes the feedback in three levels of granularities: (1) the topics with their associated sentiments, (2) the list of fine-grained features with their corresponding sentiments ordered by appearance frequency, and (3) the reviews mentioning the features. Developers and requirements analysts could navigate between the different granularities obtaining an overview of the features. This could help them decide on necessary changes to the requirements and software. Finally, since a timestamp is associated to each review, the tool could depict the trends of how features and their sentiments evolve over time and how these trends are related to specific app releases.

VI. Related Work

We focus the related work discussion in three areas: user feedback and crowdsourcing requirements, mining user feedback and app store data, as well as feature extraction in software engineering and in other domains.

A. User Feedback and Crowdsourcing Requirements

Seyff [25] and Schneider et al. [24] propose to continuously elicit user requirements with feedback from mobile devices, including information on the application context. We also previously discussed how user feedback could be considered in software development in general [14] [15]. We presented a classification of user feedback types and distinguished between "implicit feedback" and "explicit feedback". In this paper we propose an approach to systematically analyze explicit user feedback, submitted in form of informal text. Our approach helps to identify useful feedback for app analysts and developers, quantitatively evaluating the opinions about the single features, and grouping popular feature requests. We think that our approach will help the crowdsourcing of requirements.

B. Mining User Feedback

User feedback mining has recently attracted the attention of software engineering researchers resulting in several studies, most of them of exploratory nature. Harman et al. [6] analyzed technical and business aspects of apps by extracting app features from the official app descriptions using a collocation and a greedy algorithm for the extraction and grouping of features. While their feature extraction mechanism is similar to ours, the motivations are different. We are interested in extracting app features and the users sentiments associated to these features to help software teams understand the needs of their users. We therefore mine the features from the reviews and not from the semi-structured app descriptions.

Iacob and Harrison [8] extracted feature requests from app store reviews by means of linguistic rules and used LDA to group the feature requests. Our work is complementary, we are interested in extracting all app features mentioned in reviews, in presenting them in different granularity levels and extracting their associated sentiments.

Galvis Carreño and Winbladh [4] analyzed the use of LDA to summarize user review comments. Their applied model includes Sentiment Analysis, but it is not the focus of their work. Our work is complementary to theirs as we focus on describing user acceptance of features by generating different granularity levels for the extracted features and by proposing mechanisms for aggregating the sentiment on these different levels. This allows for a more detailed and focused view of user feature acceptance.

Li et al. [13] analyzed user reviews to measure user satisfaction. The authors extracted quality indicators from the reviews by matching words or phrases in the user comments with a predefined dictionary, while we use a probabilistic method (likelihood-ratio) for extracting the features. Zou et al. [29] assessed the quality of API's by analyzing user comments on the web. Unlike our approach they focused on extracting a single feature at a time instead of all features.

C. Automated Feature Extraction and Sentiment Analysis

The automatic extraction of features in text documents has been the focus of several requirements engineering researchers. Knauss et al. [11] used a Naïve Bayes approach to extract clarifications in requirements from software team communication artifacts to detect requirements that are not progressing in a project. Their approach makes use of previously tagged data, while we utilize an unsupervised approach. Dumtitru et al. [3] and Hariri et al. [5] extracted features from product descriptions to recommend feature implementation for software product lines through text mining algorithms, they then group them together through diffusive clustering. The features are mined by recognizing keywords present in bullet points lists of product descriptions, while we have no structural information indicating the presence of a feature.

While feature extraction and sentiment analysis (also called opinion mining) are relatively new in software and requirements engineering, they have been used in other domains to analyze movie reviews, for analyzing opinions of different products, such as, movies, cameras and desktop software [7], [22] and blogs [18]. Extracting features and sentiments from app stores poses different challenges than when extracting them from other product reviews, as the text in app store reviews tends to be 3 to 4 times shorter [9], having a length that is comparable to that of a Twitter message [21], but posing an additional challenge

in comparison to feature extraction in Twitter messages due to the absence of hashtags.

VII. Conclusions and Future Work

This work presents an approach for extracting app features mentioned in user reviews and their associated sentiments. The approach produces two summaries with different granularity levels. These summaries can help app analysts and developers to analyze and quantify users' opinions about the single app features and to use this information e.g., for identifying new requirements or planning future releases. We generate the summaries by combining a collocation finding algorithm, lexical sentiment analysis, and topic modeling. We obtained a precision up to 91% (59% average) and a recall up to 73% (51% average). The results show that the generated summaries are coherent and contain the most mentioned features in the reviews. We plan to enhance our approach by including linguistic rules for the detection of infrequent features. Furthermore, we plan to evaluate the approach with developers in order to measure its usefulness in industry settings and detect shortcomings. We also plan to enhance the approach with an interactive visualization. Additionally, we plan to address the limitation of the lexical sentiment analysis for detecting sarcasm and context by including the review rating in the computation of the sentiment score.

Acknowledgments

We thank Christoph Stanik for his support with the Google Play API, and Ghadeer Eresha, Safey Halim, Mathias Ellmann, Marlo Häring, Hoda Naguib, and Wolf Posdorfer for their support in the study. This work was partially supported by the Mexican Council of Science and Technology (Conacyt).

References

[1] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python*. O'reilly, 2009.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, Mar. 2003.

[3] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 181, New York, New York, USA, May 2011. ACM Press.

[4] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: an approach for software requirements evolution. In *ICSE '13 Proceedings of the 2013 International Conference on Software Engineering*, pages 582–591. IEEE Press, May 2013.

[5] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher. Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings. *IEEE Trans. Software Eng.*, 39(12):1736–1752, 2013.

[6] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. In *Proc. of Working Conference on Mining Software Repositories - MSR '12*, pages 108–111, June 2012.

[7] M. Hu and B. Liu. Mining opinion features in customer reviews. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining - KDD '04*, pages 755–760. AAAI Press, July 2004.

[8] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *MSR '13 Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 41–44. IEEE Press, May 2013.

[9] N. Jakob, S. H. Weber, M. C. Müller, and I. Gurevych. Beyond the stars: exploiting free-text user reviews to improve the accuracy of movie recommendations. In *Proceeding of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion - TSA '09*, pages 57–64, New York, New York, USA, Nov. 2009. ACM Press.

[10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document, 1990.

[11] E. Knauss, D. Damian, G. Poo-Caamano, and J. Cleland-Huang. Detecting and classifying patterns of requirements clarifications. In *Requirements Engineering Conference (RE), 2012 20th IEEE International*, pages 251–260, Sept. 2012.

[12] O. Kucuktunc, B. B. Cambazoglu, I. Weber, and H. Ferhatosmanoglu. A large-scale sentiment analysis for Yahoo! Answers. In *Proc. of the International conference on Web search and data mining - WSDM '12*, pages 633–642, Feb. 2012.

[13] H. Li, L. Zhang, L. Zhang, and J. Shen. A user satisfaction analysis approach for software evolution. In *Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on*, volume 2, pages 1093–1097. IEEE, 2010.

[14] W. Maalej, H.-J. Happel, and A. Rashid. When users become collaborators. In *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA '09*, page 981, New York, New York, USA, Oct. 2009. ACM Press.

[15] W. Maalej and D. Pagano. On the Socialness of Software. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 864–871. IEEE, Dec. 2011.

[16] W. Maalej and M. P. Robillard. Patterns of Knowledge in API Reference Documentation. *IEEE Transactions on Software Engineering*, 39(9):1264–1282, 2013.

[17] H. Manning, Christopher D., Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[18] Q. Mei, X. Ling, M. Wondra, H. Su, and C. Zhai. Topic sentiment mixture: modeling facets and opinions in Weblogs. In *Proc. of the 16th international conference on World Wide Web - WWW '07*, pages 171–180, May 2007.

[19] G. A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[20] K. A. Neuendorf. *The Content Analysis Guidebook*. Sage Publications, 2002.

[21] D. Pagano and W. Maalej. User feedback in the appstore : an empirical study. In *Proc. of the International Conference on Requirements Engineering - RE '13*, pages 125–134, 2013.

[22] A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing - HLT '05*, pages 339–346. Association for Computational Linguistics, Oct. 2005.

[23] R. L. Rosnow. *Beginning behavioral research: a conceptual primer*. Pearson/Prentice Hall, Upper Saddle River, {N.J}, 6th ed edition, 2008.

[24] K. Schneider, S. Meyer, M. Peters, F. Schliephacke, J. Mörschbach, and L. Aguirre. *Product-Focused Software Process Improvement*, volume 6156 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, June 2010.

[25] N. Seyff, F. Graf, and N. Maiden. Using mobile re tools to give end-users their own voice. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 37–46. IEEE, 2010.

[26] J. Shao, Y. Wang, X. Deng, S. Wang, et al. Sparse linear discriminant analysis by thresholding for high dimensional data. *The Annals of statistics*, 39(2):1241–1265, 2011.

[27] M. Thelwall, K. Buckley, and G. Paltoglou. Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63(1):163–173, Jan. 2012.

[28] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558, Dec. 2010.

[29] Y. Zou, C. Liu, Y. Jin, and B. Xie. Assessing Software Quality through Web Comment Search and Analysis. In *Safe and Secure Software Reuse*, pages 208–223. Springer, 2013.