Using App Reviews for Competitive Analysis: Tool Support

Faiz Ali Shah*
Institute of Computer Science
University of Tartu
Tartu, Estonia
faiz.ali.shah@ut.ee

Kairit Sirts
Institute of Computer Science
University of Tartu
Tartu, Estonia
kairit.sirts@ut.ee

Dietmar Pfahl
Institute of Computer Science
University of Tartu
Tartu, Estonia
dietmar.pfahl@ut.ee

ABSTRACT

Play Store and App Store have a large number of apps that are in competition as they share a fair amount of common features. User reviews of apps contain important information such as *feature evaluation*, *bug report* and *feature request*, which is useful input for the improvement of app quality. Automatic extraction and summarization of this information could offer app developers opportunities for understanding the strengths and weaknesses of their app and/or prioritizing the app features for the next release cycle. To support these goals, we developed the tool REVSUM which automatically identifies developer-relevant information from reviews, such as reported bugs or requested features. Then, app features are extracted automatically from these reviews using the recently proposed rule based approach *SAFE*. Finally, a summary is generated that supports the application of the following three use cases: (1) view users' sentiments about app features in competing apps, (2) detect which summarized app features were mentioned in bug related reviews, and (3) identify new app features requested by users.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; *Software evolution*; Software maintenance tools.

KEYWORDS

App review analysis, competitive analysis, app feature extraction, app review classification, sentiment analysis

ACM Reference Format:
Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. 2019. Using App Reviews for Competitive Analysis: Tool Support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics (WAMA '19), August 27, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3340496.3342756

1 INTRODUCTION

App marketplaces such as Play Store and App Store host a large number of competing apps, i.e. apps sharing a fair set of common features [12]. These apps are fiercely competing with each other for profit which is directly linked to the number of downloads and user ratings. In such a highly competitive environment, the data buried in the user reviews such as reported bugs, requested features or evaluation of existing app features is a useful information to app developers for enhancing the quality of their apps [5, 10]. However, especially popular apps receive a large number of user reviews every day making the manual analysis of user reviews unfeasible. Therefore, prior studies [1, 4] have adopted techniques from Natural Language Processing (NLP) research that help to automatically extract and summarize useful information from user reviews.

Some previous studies have adopted LDA-based topic models to summarize useful information automatically extracted from user reviews [1, 4]. Few studies [5, 6] have focused on summarizing reviews at the level of granularity of app features. In these studies, automatic analysis of user reviews was performed on a single app. Similar to this work, the studies of Shah et al. [12] and Dalpiaz et al. [3] aimed at performing a more fine-grained analysis of user reviews on multiple apps and specifically with the objective of competitive analysis.

The tools developed by Shah et al. [12] and Dalpiaz et al. [3] extract app features directly from full review texts. However, not all review sentences contain information that is relevant for app developers (ADs). Figure 1 shows an example review where the relevant sentence containing useful information about an app feature is marked in blue and irrelevant sentences expressing additional sentiment tone but not much extra information are marked in pink. Filtering out irrelevant review sentences may improve the automatic extraction of app features from user reviews as this could reduce the number of false positives (FPs)[1] in the set of automatically extracted app features.

"I would like the ability to share a complete folder not just one file at a time.I'm not sure if I'm just not seeing this feature?It seems like a basic function at this point, since you can do basically everything else within the app platform. Other than that, I am very happy with the experience."

Figure 1: Example of a review text containing relevant (blue color) and irrelevant (pink colour) review sentences.

We present a tool named RevSum[2] that automatically analyses user reviews of a reference app in comparison with a set of competing apps to help the developers of the reference app improve the quality their app. Different from previous tools by Dalpiaz et al. [3] and Shah et al. [12], RevSum first classifies review sentences into the five types *feature evaluation* (E), *bug report* (B), *feature request* (R), *praise* (P), and *other* (O), and then automatically extracts app features from review sentences of types E, B, and R that we

[1] FP is an app feature automatically extracted from a review text which turns out not to be a true app feature.
[2] the name was derived as an acronym from the phrase Review Summary

WAMA '19, August 27, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-6858-2/19/08…$15.00
https://doi.org/10.1145/3340496.3342756

believe contain the most relevant information for ADs. The tool SURMiner [5] classifies review sentences into the same classes as RevSum but it only presents the feature-level sentiment summary of review sentences belonging to type E containing feature evaluations. Additionally, our tool RevSum also generates feature-level summaries of the reported bugs as well as new features requested by users. Moreover, our tool also supports functionality by which app developers can compare the features of their own app with the same features of competitor apps.

Another difference between our tool RevSum and the previous approaches lies in the method of extracting app features from user reviews. While Shah et al. [12], Dalpiaz and Parente [3], Guzman and Maalej [6] used a collocation algorithm for app feature extraction, we adopt the recently proposed rule-based approach *SAFE* [8] for this purpose. The *SAFE* approach relies on a small set of part-of-speech (POS) patterns for extracting app features thus avoiding the use of complex linguistic tools [5, 11] and supervised machine-learning approaches. Although *SAFE* is more restrictive in extracting app features than the collocation extraction method, it still suffers from relatively low precision [15] due to a high FP rate. RevSum provides several options to overcome the noise stemming from low precision of SAFE: (1) tool user has the ability to filter *SAFE* extracted features by frequency threshold, (2) the precision of *SAFE* is higher on features extracted from app descriptions compared to features extracted from user reviews; therefore, the tool user is given an option to choose the app description as an alternate source to extract app features. In this scenario, the app features extracted from app description will be searched in the review sentences for further analysis. (3) RevSum also enables users to manually revise the app features extracted by *SAFE* from the app description.

In summary, our contribution in this paper is the development of the tool RevSum. Different to existing tools, RevSum analyzes developer-relevant information in app reviews at the level of app features and supports the following three main use cases (UCs) for comparing a reference app with its competitor app:

UC 1: Summarize users' sentiments towards app features.
UC 2: Summarize app features mentioned in bug related reviews.
UC 3: Summarize new app features requested by app users.

The rest of the paper is structured as follows. Section 2 summarizes the related work. In Section 3, we describe the technical approach used to analyze the user reviews. In Section 4, the description of use cases supported by our tool REVSUM is presented. Section 5 demonstrates the application of these use cases. Section 6 provides a discussion. In Section 7, threats to validity are examined. Conclusions are presented in Section 8.

2 RELATED WORK

In this section, we summarize the work related to our study. We categorize the existing research of app review analysis into three main topics: (a) review summarization, (b) automatic app feature extraction, and (c) competitive analysis. Our study mainly belongs to the topic of competitive analysis but also addresses other topics.

2.1 Review Summarization

SUR-Miner system[5] first classifies review sentences into the following five categories: *feature evaluation*, *bug report*, *feature request*, *praise* and *other* and then performs feature-level sentiment analysis on review sentences belonging to type *feature evaluation*. The first step of our tool is similar to SUR-Miner but besides performing feature-level analysis of *feature evaluation* sentences, we also analyse review sentences of type *bug report* and *feature request*.

Dąbrowski et al. [2] proposed a tool which takes as input a set of user-inserted app features and extracts the reviews where these features are mentioned. Then, the extracted reviews are automatically classified into the following three classes: evaluating an app feature positively, negatively or neutrally, reporting a bug or asking for a feature enhancement. Finally, the categorized reviews are visualized on the tool's dashboard in support of requirements prioritization.

Guzman and Maalej [6] performed feature level analysis on the reviews of a single app by extracting app features using the collocation algorithm and then grouping the extracted features using topic modeling. Chen et al. [1] proposed the system AR-Miner that classifies review sentences into two classes: *informative* and *non-informative*. Then, LDA topic modeling is applied on the review sentences classified as *informative*. The recently developed tool SURF [4] summarizes the improvements requested by app users. The tool first classifies review sentences into intention categories (i.e. *information giving*, *information seeking*, *feature request*, *problem discovery*, and *other*) and then clusters these review sentences into topics using an automatic topic classifier. Different from these studies, our tool uses the taxonomy of Gu and Kim [5] to classify review sentences, and then the review sentences of the relevant categories (i.e., *feature evaluation*, *bug report* and *feature request*) are summarized at the level of granularity of app features.

2.2 App Feature Extraction

Johann et al. [8] proposed a rule-based approach called *SAFE* for automatically extracting features from user reviews and app descriptions. Authors of *SAFE* reported that its performance is better than the topic modeling approach proposed by Guzman et al. [6] and collocation algorithm used by Harman et al. [7]. Although our previous replication study that evaluated the *SAFE* performance on different review datasets found that the average performance of *SAFE* is lower than reported in the original study [15], we use our own implementation of *SAFE* in our tool because of its simplicity, additionally adopting several techniques to filter out noisy app features extracted by *SAFE* rules. Our recent study[14] also assessed the performance of supervised machine learning models for extracting features from app reviews as supervised methods have shown to perform well on the reviews of restaurant and laptop domain. However, we found that on app reviews, the performance of supervised learning models is low, especially in terms of recall.

Gu and Kim [5] used twenty-six manually designed templates to extract app features from review sentences of type *feature evaluation*. Keertipati et al. [9] adopted a simple heuristic that extracts all nouns as candidate app features. Vu et al. [16] extracted all potential keywords from user reviews and ranked them by the review rating and their occurrence frequency. In all these studies, the extracted app features were not evaluated against the set of manually

labeled reviews. Therefore, the performances of these approaches for extracting app features from reviews is not known.

2.3 Competitive Analysis

Our previous tool [12] analyzed app reviews to perform feature-level comparison of competing apps. Recently, Dalpiaz and Parente [3] presented a tool that generates requirements by performing a SWOT (Strengths, Weaknesses, Opportunities, and Threats) analysis on the reviews of a reference app in comparison to its competitor apps. Unlike the proposed tool RevSum, these tools analyse all review sentences and do not filter out irrelevant sentences before extracting app features. Therefore, the extracted app features in these tools may contain a large number of FPs.

3 APPROACH

As shown in Figure 2, our proposed tool RevSum comprises of the following five main components: (a) Apps selection, (b) Review collection, (c) Review classification and feature extraction, (d) Feature level analysis of the selected apps, and (e) Visualization. The order in which the steps are performed in each component are numbered in Figure 2. In the following subsections, the functionality of each component is explained in detail.

3.1 Apps Selection

This component collects the names of the reference app and its competitor apps that the tool user[3] wants to analyze. In step 1 (see Figure 2), the user inputs the name of the reference app and based on this input, the system invokes a call to the RESTful API[4] (step 2) which retrieves a list of similar apps (steps 3 and 4) from Google Play Store and presents them to the user (step 5). In the final step 6, the user selects the desired competitor apps from the retrieved list. Further analyses are performed as comparisons between the reference and the selected apps.

3.2 Review Data Collection

This component collects user reviews of the reference app and its competitor apps selected in the previous component (i.e. Apps selection). For each selected app, the sub-component "collect reviews" fetches the most recent 400 reviews, calling the RESTful API[4], and saves them in the database for later use.

3.3 Review Classification and Feature Extraction

This component is responsible for finding information from app user reviews relevant for the tool user. First, review sentences are classified, then app features are automatically extracted.

To automatically classify review sentences, user reviews collected by the previous component are retrieved from the database (step 1). Then, each review is segmented into sentences (step 2) using Stanford CoreNLP library[5]. In the pre-processing step 3, the review sentences are cleaned by replacing some typos and contractions. For this, we use the list[6] of words proposed by Gu and Kim [5]. In step 4, the simple *bag-of-word* classification model of Shah et al. [13] is used to classify review sentences into the following five types: *feature evaluation* (E), *bug report* (B), *feature request* (R), *praise* (P) and *other* (O). The analysis proceeds with the sentence types E, B and R. Sentences of types P and O are discarded.

Steps 5, 6 and 7 extract app features from categorized review sentences using the rule-base *SAFE* method. The last step 8 clusters the extracted feature terms by stemming, i.e. by reducing each word into to their root form. For this task we used the SnowballStemmer[7] available in NLTK library. As a result, feature terms such as *uploading pictures*, *uploads picture*, *uploaded pictures* are mapped onto the same feature term *upload picture*.

The performance of the *SAFE* method improves when app features are extracted from app descriptions and not from user reviews. Therefore, in our tool, we also allow the user to select app description as a source for extracting app features instead of user reviews.

3.4 App Feature Level Analysis

This component performs feature-wise analysis on the review sentences of the reference app and its competitor apps. This analysis is performed separately for sentences of type E, B and R (steps 1, 4 and 6, respectively).

Step 2 performs sentiment analysis on review sentences of type E containing feature evaluations using Standford CoreNLP Library[5]. For an input review sentence, the CoreNLP library outputs the sentiment score between [0,4] where 0 means "very negative" and 4 means "very positive", and 2 denotes "neural". For better readability, we scale the sentiment score to the range [-2, +2]; now -2 means "very negative", +2 represents "very positive" and 0 is "neutral". Similar to Gu and Kim [5], we also used review ratings to improve the accuracy of the sentiment analysis adjusting the sentiment score in case where the score predicted by the sentiment analyser and the review rating score fully contradict each other. For instance, if the review rating is 5 or 4 but the predicted sentiment score is -1 or -2 then the sentiment score is adjusted by +1. Similarly, if the review rating is 1 or 2 and the predicted sentiment score is +2 or +1 then the sentiment score is adjusted by -1.

Given the set of app features $F = \{f_1, f_2, ..., f_n\}$ mentioned in app review sentences $R = \{r_1, r_2, ..., r_m\}$, the average sentiment score FS_i of the app feature f_i is calculated using Equation (1):

$$FS_i = \frac{1}{C(f_i)} \sum_{j=1}^{m} S_{i,j}, \tag{1}$$

where, $S_{i,j}$ is the sentiment score of the sentence r_j mentioning feature f_i and $C(f_i)$ is the total number of review sentences mentioning the feature f_i. In other words, the sentiment score FS_i of the app feature f_i is the average sentiment score of the review sentences that mention this feature.

[3] In the following, we use the term *user* to refer to the user of the tool and not to the end user of the app itself.
[4] https://github.com/facundoolano/google-play-scraper
[5] https://stanfordnlp.github.io/CoreNLP/
[6] https://guxd.github.io/srminer/appendix.html
[7] https://www.nltk.org/_modules/nltk/stem/snowball.html

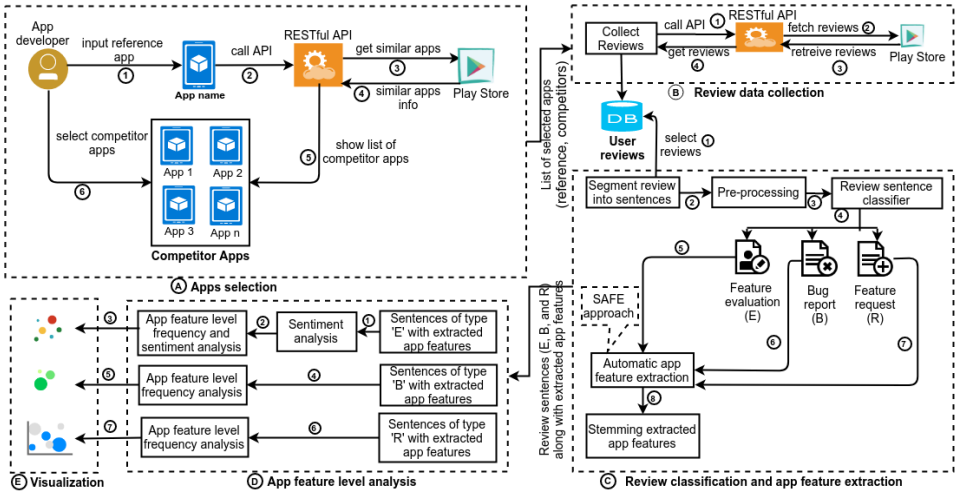

Figure 2: RevSum approach for summarizing app reviews at level of app features .

3.5 Visualization

This component visualizes the feature-wise review analyses of the reference app and its competitor apps. For review sentences containing feature evaluations (type E), the plot displays app features on the horizontal axis and their corresponding sentiment scores as app icons on the vertical axis (step 3). The size of the icons corresponds to the frequency of each app feature in the reviews. For review sentences categorized as bug reports and feature requests (types B and R), the vertical axis of the plot displays the frequency of each app feature for each app (steps 5 and 7).

4 USE CASES

This section presents three main use cases that are implemented in RevSum to help app developers to improve their apps.

UC 1 - The usage of the UC 1 is for app developers to compare their app (i.e. the reference app) with other competitor apps by monitoring the users' sentiments conveyed towards app features. Table 1 describes the main and alternative flow of this use case.

UC 2 - The primary usage of UC 2 is to visualize the app features mentioned in bug related reviews of the selected apps. The description of the Use Case 2 is presented in Table 2. The flow of alternate use cases (i.e., [A1][A2]) is the same as described in UC 1; therefore, they are not repeated here.

UC 3 - The usage of UC 3 is to view which new app features were requested by the app users in the reviews of the selected apps. The description of the UC 3 is shown in Table 3. The flow of alternate use cases (i.e., [A1]) is the same as described in UC 1; thus, they are not repeated here.

5 APPLICATION OF THE USE CASES

This section demonstrates the application of the three use cases described in Section 4. Due to space limitation, we only demonstrate the normal flow of these use cases but the tool RevSum and its manual[8] are available online.[9] The pre-condition for all three use cases is that the tool user has already selected the reference app and the competitor apps. For demonstration purposes, we selected Nike Run club as the reference app and Run keeper GPS Running and Map my Ride as its competitor apps. In the following sub-sections, we demonstrate the application of each use case.

5.1 Use Case 1

As described in Section 4, the purpose of the UC 1 is to view users' sentiments (i.e., positive, negative, or neutral) towards app features of the reference app and its competitor apps. The approach used to extract this information from user reviews is described in detail in Section 3. For the selected apps Nike Run club, Map my Ride, and Run keeper, the top plots (labeled as "a" and "b") in Figure 3 shows the output of UC 1. The plot, labeled as "a", shows for each extracted app feature its average sentiment score. Each app is represented by its icon and the size the icon corresponds to the frequency of the app feature in the reviews of this app. Feature-level sentiment summary in the top-left plot of Figure 3 automatically generated from user reviews can help app developers to understand the strengths and weaknesses of their app. For instance, the app feature "voic feedback" is mentioned more frequently in the reviews of the reference app Nike Run Club compared to the competitor apps Run keeper and Map my Ride. However, the average sentiment score of this feature is lower for the reference app, reaching to the negative side of the sentiment scale, whereas for the competitor apps, the average sentiment score is higher and in the positive range. Based on that information, developers of the app Nike Run club can look in more detail what the app users do not like about this particular feature in the Nike Run club and what they do like about it in the competitor apps. Clicking on the icon of the respective app lists the review sentences mentioning this app feature together with their sentiment scores. A sample view of the output is shown on the top right of Figure 3 (labeled as "b"). Inspecting these sentences can help to find out how to improve the app feature "voic feedback" to remain competitive in the app market.

[8] http://bit.ly/tool_manual
[9] http://18.219.206.183:8088/

Table 1: Use Case 1

Role	App Developer (AD)
Description	View users' sentiments towards the app features of competitor apps
Trigger	AD is interested in finding the strengths and weakness of his app.
Pre-conditions	AD has already selected the reference app and its competitor apps
Post-conditions	AD views users' sentiments towards the app features of competitor apps.
Normal flow	(1) AD selects the sentence type category *feature evaluation* from the list of sentence types. (2) For the selected apps, AD sees the list of app features extracted from the review sentences of type 'E' with their frequencies (the default threshold for app feature frequency>=5) **[A1][A2]** (3) For the selected apps, AD sees the plot showing the app-features on x-axis and users' sentiments and feature frequency distribution on y-axis.
Alternate flow	**A1.** AD changes the default frequency threshold for app features automatically extracted from review sentences: (1) AD changes the frequency threshold for app features using the slider control. (2) AD sees the updated list of app features fulfilling the selected frequency threshold criteria. (3) For the selected apps, PD views the plot showing users' sentiments towards app features. **A2.** AD selects APP DESCRIPTION as a source for extracting app features: (1) AD sees the list of app features extracted from app description using SAFE approach. (2) AD sees the list of app features fulfilling the frequency threshold criteria. (3) AD manually revises the list of extracted features either by annotating new app features in app description or removing the FPs extracted from app description. (4) For the selected apps, AD sees the plot showing users' sentiments conveyed on app features.

Table 2: Use Case 2

Role	App Developer (AD)
Description	Show which app features were mentioned in bug related reviews.
Trigger	AD is interested in finding out the app features mentioned in bug related reviews.
Pre-conditions	AD has already selected his app and competitor apps .
Post-conditions	For the selected apps, AD views the app features mentioned in bug related reviews.
Normal flow	(1) AD selects the sentence type category BUG REPORT from the list of sentence types. (2) AD sees the list of app features extracted from review sentences of the selected apps with their frequencies (default threshold for app feature frequency >=2) **[A1][A2]** (3) For the selected apps, AD sees the plot showing app-features on the x-axis and the frequency of each app feature is shown on the y-axis.

Table 3: Use Case 3

Role	App Developer (AD)
Description	View which new app features were requested by users in the reviews of competitor apps.
Trigger	AD wants to find out new features requested by users.
Pre-conditions	AD has already selected his app and its competitor apps.
Post-conditions	AD views the summary of newly requested features in the reviews of competitor apps.
Normal flow	(1) AD selects the sentence type category FEATURE REQUEST from the list of sentence types. (2) AD sees the list of app features extracted from review sentences of the selected apps along with their frequencies (default threshold for app feature frequency >=2) **[A1]** (3) For the selected apps, AD sees the plot showing newly requested features on the x-axis and the frequency of each app feature on the y-axis.

5.2 Use Case 2

Experiencing bugs in apps are the main cause of frustration for app users. UC 2 enables app developers to find app features that were mentioned in the review sentences classified as *bug reports*. The middle plot in Figure 3 (labeled as "c") shows an example output of the UC 2, where extracted app features are shown on the horizontal axis while the frequency of these app features are shown on the vertical axis. As can be seen from Figure 3, the app feature "gps signal" is mentioned approximately 15 times in the NIKE RUN CLUB app and its competitor app RUN KEEPER. The frequent mentioning of this app feature in bug-related review sentences hints that there might be something that needs to be fixing about this feature. Similar to the Use case 1, by clicking on the app icon corresponding to the app feature, the tool user can read all bug-related review sentences where this particular app feature was mentioned. A sample view of it is shown in the middle-right plot of Figure 3 (labeled as "d").

5.3 Use Case 3

App reviews can also contain information about new features requested by users. Extracting and using this information to guide decisions about what to include in next releases can improve the competitiveness of an app. The goal of UC 3 is to summarize the information about new requested features automatically extracted from review sentences classified as *feature requests*. These extracted app features are shown to the tool user along with their frequencies. An example output of UC 3 is shown in the bottom-left plot

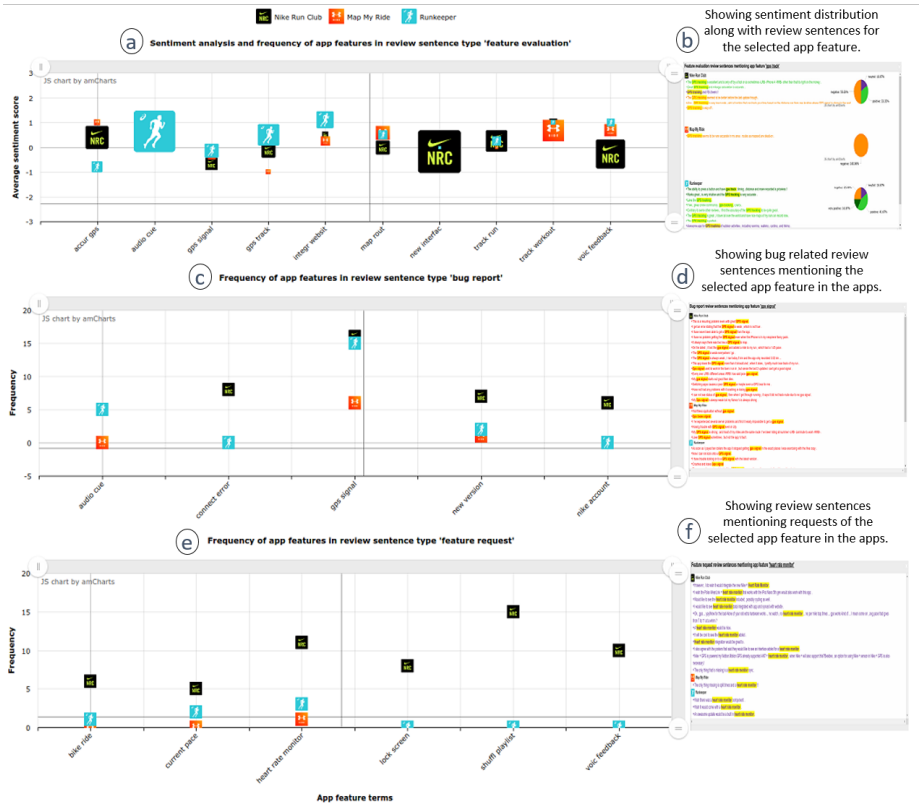

Figure 3: Demonstration of all three use cases

of Figure 3(labeled as "e"). The extracted app features are shown on the x-axis and the frequency with which they occur in each app is shown on the y-axis. High frequency of requests for an app feature could be a hint for the AD to prioritize this particular app feature for the next release cycle. For instance, the app features "shuffle playlist" and "voice feedback" are requested by app users in the example reference app Nike Run Club at least ten times, making them good candidates for inclusion in the next release cycle. Requests for these features were not found in the reviews of the competitor apps but this does not tell anything about the presence or absence of these features in the competitor apps. However, the user can proceed with UC 1 and search for these features and their evaluations from the user reviews of competitor apps.

Similar to UC 1 and UC 2, the tool user can read all review sentences where a particular app feature was requested by clicking on the app icon corresponding to that app feature. A sample output of it is shown in the bottom-right plot of Figure 3 (labeled as "f").

6 DISCUSSION

The usefulness of RevSum mainly depends on the accuracy of three components: (a) the classification model that automatically categorizes review sentences, (b) the approach used for extracting app features automatically from review sentences and (c) the sentiment analysis tool that predicts the sentiment of review sentences.

Since the reviews of any app can be input to our tool for analysis, the model used for review sentence classification needs to be app agnostic. The model we use for our tool is trained and validated on the labeled review sentences of all apps in the Gu and Kim [5] dataset. Based on the results reported in our study [13], the model can categorize the review sentences of types *feature evaluation*, *bug report*, and *feature request* with a precision of 75.4%, 67.1%, 63.4% and recall of 68.1%, 68.9%, 71.0%, respectively. Although there is room for improvement in the classifier's performance, the average recall close to 70% indicates that our model could still be useful for finding relevant information from a large collection of user reviews.

We use the rule-based approach *SAFE* for extracting app features automatically from user reviews. In our recent replication study [15], *SAFE* had an average precision of 12% and recall of 54% on different review datasets, which objectively is quite low. However, our previous experiments with supervised machine-learning methods were even less encouraging when employed for app feature extraction from user reviews[14]. Therefore, we opted for SAFE approach, mostly because of its simplicity. To improve its precision, we allow tool users to filter *SAFE* extracted app features by frequency threshold. For instance, by setting the threshold to a higher frequency will show only those app features that are mentioned frequently in user reviews and thus reduces the number of FPs. The review classification step which is applied before app feature extraction filters out irrelevant review sentences and that also helps

to reduce the number of FPs extracted by the *SAFE* method. Finally, in order to improve precision at the cost of lower recall, the tool offers an option to extract app features from the app description instead of user reviews. In this case, the tool user also has a choice to revise the extracted list of app features by manually annotating them in the app description or by removing the false app features incorrectly extracted from the app description.

For predicting the sentiment scores of review sentences, we used the library available in the Stanford CoreNLP tool[5] that has a reported accuracy[10] of 80%. App reviews are outside its domain because the model is trained and evaluated on movie reviews[10]. To improve the sentiment analysis accuracy on out-of-domain app review data, following Gu and Kim [5], we used the review ratings to adjust the predicted sentiment score when the score predicted by the CoreNLP fully contradicts with the review rating score (see Section 3.4). To estimate our tool's accuracy for the sentiment prediction task, two persons[11] manually and independently assigned sentiment labels (i.e., *positive*, *negative*, and *neutral*) to 100 randomly selected review sentences. There were only four cases where the two annotators assigned conflicting labels and these cases were resolved after discussion. Then, the same review sentences were input to our tool for sentiment prediction. Using human labels as ground truth, the estimated accuracy of the tool was 71%. We noticed that most of the 19% of the wrongly predicted cases were because of the tool's confusion between the *neutral* and *positive* classes.

7 THREATS TO VALIDITY

The use cases presented in this paper were not proposed by ADs but by the authors attempting to put themselves into the shoes of ADs. Therefore, the use cases presented in this paper may not represent the real viewpoint of ADs. However, to mitigate this threat, we discussed these use cases with few ADs who confirmed that these use cases make sense to them. Moreover, we currently design a survey to more thoroughly study the usefulness of the tool for ADs.

We hypothesize that classifying review texts into five sentence types corresponding to *feature evaluation*, *bug report*, *feature request*, *praise* and *other* and filtering out sentences of type *praise* and *other* helps to get rid of irrelevant information and thus improves the precision of app feature extraction with the *SAFE* method. However, this assumption still needs to be confirmed.

To find competing apps similar to the reference app, we rely on the Play Store API[4] not knowing the exact algorithm how this API finds the set of similar apps and to what extent the results returned by the API are correct.

8 CONCLUSION

App reviews contain useful information that potentially helps ADs enhance the quality of their app. We presented a tool for competitive analysis of several apps based on the information extracted from app user reviews. Our tool RevSum first automatically categorizes review sentences discarding irrelevant sentences that do not contain any information about app features, and then analyzes the review sentences of each competing app at app feature level.

RevSum currently implements three use cases: **UC 1** - view users' sentiments towards app features, **UC 2** - view which app features were mentioned in bug related reviews, and **UC 3** - view which new app features were requested by users. These use cases can help app developers to analyze the strengths and weaknesses of their app and/or to prioritize app features for next release cycles. The main limitation of our study is that the usefulness of these use cases has not been evaluated by real app developers yet. However, a survey evaluating the usefulness of the presented tool RevSum for the app developers is currently being designed.

ACKNOWLEDGMENTS

This research was supported by the institutional research grant IUT20-55 of the Estonian Research Council and the Estonian Center of Excellence in ICT research (EXCITE).

REFERENCES

[1] Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In *Proceedings of the 36th Int'l Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 767–778. https://doi.org/10.1145/2568225.2568263

[2] Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. 2019. Finding and Analyzing App Reviews Related to Specific Features: A Research Preview. In *Int'l Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 183–189. https://doi.org/10.1007/978-3-030-15538-4

[3] Fabiano Dalpiaz and Micaela Parente. 2019. RE-SWOT: From User Feedback to Requirements via Competitor Analysis. In *Int'l Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 55–70.

[4] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Corrado A. Visaggio, and Gerardo Canfora. 2017. SURF: Summarizer of User Reviews Feedback. In *39th Int'l Conference on Software Engineering Companion (ICSE-C)*. IEEE, 55–58.

[5] Xiaodong Gu and Sunghun Kim. 2015. "What Parts of Your Apps are Loved by Users?". In *30th Int'l Conference on Automated Software Engineering*. IEEE, 760–770. https://doi.org/10.1109/ASE.2015.57

[6] Emitza Guzman and Walid Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *22nd Int'l Requirements Engineering Conference (RE)*. IEEE, 153–162.

[7] Mark Harman, Yue Jia, and Yuanyuan Zhang. 2012. App store mining and analysis: MSR for app stores. In *9th Working Conference on Mining Software Repositories (MSR)*. IEEE, 108–111.

[8] Timo Johann, Christoph Stanik, Walid Maalej, et al. 2017. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In *25th Int'l Requirements Engineering Conference (RE)*. IEEE, 21–30.

[9] Swetha Keertipati, Bastin T. R. Savarimuthu, and Sherlock A. Licorish. 2016. Approaches for Prioritizing Feature Improvements Extracted from App Reviews. In *Proceedings of the 20th Int'l Conference on Evaluation and Assessment in Software Engineering (EASE '16)*. ACM, New York, NY, USA, Article 33, 6 pages.

[10] Dennis Pagano and Walid Maalej. 2013. User feedback in the appstore: An empirical study. In *21st Int'l Requirements Engineering Conference (RE)*. IEEE, 125–134. https://doi.org/10.1109/RE.2013.6636712

[11] Guang Qiu, Bing Liu, Jiajun Bu, and Chun Chen. 2011. Opinion Word Expansion and Target Extraction Through Double Propagation. *Comput. Linguist.* 37, 1 (March 2011), 9–27. https://doi.org/10.1162/coli_a_00034

[12] Faiz A. Shah, Yevhenii Sabanin, and Dietmar Pfahl. 2016. Feature-based Evaluation of Competing Apps. In *Proceedings of the Int'l Workshop on App Market Analytics (WAMA 2016)*. ACM, New York, NY, USA, 15–21.

[13] Faiz A. Shah, Kairit Sirts, and Dietmar Pfahl. 2018. Simple App Review Classification with Only Lexical Features. In *Proceedings of the 13th Int'l Conference on Software Technologies*. SciTePress, 112–119.

[14] Faiz A. Shah, Kairit Sirts, and Dietmar Pfahl. 2018. Simulating the Impact of Annotation Guidelines and Annotated Data on Extracting App Features from App Reviews. In *Proceedings of the 14th Int'l Conference on Software Technologies*. SciTePress (appear in print), 112–119.

[15] Faiz A. Shah, Kairit Sirts, and Dietmar Pfahl. 2019. Is the SAFE Approach Too Simple for App Feature Extraction? A Replication Study. In *Int'l Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 21–36. https://doi.org/10.1007/978-3-030-15538-4

[16] Phong M. Vu, Tam T. Nguyen, Hung V. Pham, and Tung T. Nguyen. 2015. Mining User Opinions in Mobile App Reviews: A Keyword-Based Approach. In *30th Int'l Conference on Automated Software Engineering (ASE)*. IEEE, 749–759.

[10] https://www.aclweb.org/anthology/D13-1170

[11] One of them was the main author of this paper.