Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews

Claudia Iacob, Rachel Harrison
Department of Computing and Communication Technologies
Oxford Brookes University
Oxford, United Kingdom
{iacob, rachel.harrison}@brookes.ac.uk

***Abstract*—Mobile app reviews are valuable repositories of ideas coming directly from app users. Such ideas span various topics, and in this paper we show that 23.3% of them represent feature requests, i.e. comments through which users either suggest new features for an app or express preferences for the re-design of already existing features of an app. One of the challenges app developers face when trying to make use of such feedback is the massive amount of available reviews. This makes it difficult to identify specific topics and recurring trends across reviews. Through this work, we aim to support such processes by designing MARA (Mobile App Review Analyzer), a prototype for automatic retrieval of mobile app feature requests from online reviews. The design of the prototype is a) informed by an investigation of the ways users express feature requests through reviews, b) developed around a set of pre-defined linguistic rules, and c) evaluated on a large sample of online reviews. The results of the evaluation were further analyzed using Latent Dirichlet Allocation for identifying common topics across feature requests, and the results of this analysis are reported in this paper.**

***Index Terms*—Online reviews, mobile apps, feature requests.**

I. Introduction

Online reviews are an instrument for facilitating both customers' purchase decisions and product evaluations [1, 2]. Mobile app reviews in particular, are valuable repositories of ideas and comments coming directly from users and benefiting both other users and app developers. On the one hand, users may look for comments on the app's usability and potential bugs before deciding whether to purchase/install an app or not based on other users' experience with the app. On the other hand, for developers, reviews are evaluations which may include ideas for improvement or feature requests. We consider feature requests to be comments through which users either suggest new features for an app or express preferences for the re-design of features which already exist. Previous work has focused on extracting feature requests information from the text of app store descriptions [4] and understanding software reuse in the Android mobile app market along reuse by inheritance and class reuse [5].

Through this work we investigate the degree to which app users use online reviews to express feature requests and possible ways to automate the process of mining for such requests in reviews available online. There are several challenges involved. First, the number of reviews associated with one app often exceeds a human's capacity to read them all in order to identify recurring issues and trends. The Facebook app alone has almost 6 million reviews. Moreover, the most popular non free apps on the Google App store get an average of 45.5 reviews per day. Second, reviews have a style of their own; they are usually short, unstructured and seldom obey grammar and punctuation rules. Moreover, users express opinions in unconventional ways, often using unconventional syntax or sarcasm to vent their feelings after purchasing an app. Even though humans can usually interpret such reviews, it is challenging to automate such interpretations [6].

We first considered a sample of reviews for analysis and we aimed to identify how much of the users' feedback consists of feature requests and how these feature requests are expressed through reviews. Such findings informed the design of a prototype system able to identify and retrieve feature requests from online reviews of mobile apps. We evaluated the prototype using a large sample of reviews and we used Latent Dirichlet Allocations [7] to identify common topics across the feature requests obtained as a result of this evaluation. The paper is structured as follows: Section 2 reports on the online review analysis, Section 3 describes the prototype designed for the automation retrieval of mobile app feature requests from online reviews, and Section 4 presents the LDA analysis performed on the feature requests retrieved during the prototype's evaluation. Lastly, the paper ends with conclusions and ideas for future iterations of the prototype.

II. Feature Request in Mobile App Reviews

In our attempt to understand how much feedback consists of feature requests, we selected the top 6 most popular categories from Google app store and, for each category, we randomly generated a number n_k of app identifiers, where n_k is the size of a sample taken from the population of apps in that category, directly proportional to the total number of apps in that category. The categories selected and the numbers of apps for each category are: Personalization (54 apps), Tools (14 apps), Books and References (45 apps), Education (27 apps), Productivity (19 apps), and Health & Fitness (10 apps). For each randomly selected app, we automatically extracted and stored the reviews provided for it by users. For each review, we automatically collected the date it was posted, the rating the user gave, the device associated with it, and the version of the app, as well as the title and the text of the actual review. Out of the 169 apps randomly selected, 8 apps had no reviews assigned to them which left us with 161 reviewed apps and a

total of 3279 reviews. The average rating of the 161 apps considered is 4.27 (on a scale of 1 to 5), while the average price per app is £1.92.

We went through the sample of reviews and manually extracted those fragments of reviews which expressed feature requests (eg. "*you need to add an exit button*", "*wish it had a math language pack*", "*wish we could have icons next to the counters*"). We found that reviews were extensively used to provide developers with details about features that could be improved or added to the apps. This behavior was observed in 23.3% of the feedback analyzed: 763 reviews out of the 3279 considered expressed feature requests. Apps rated higher get far more feature requests than apps rated lower. Only 37.5% of the apps rated lower than 3 were associated with feature requests, whereas more that 70% of the apps rated higher than 3 were associated with at least one feature request. Price wise, there was no significant difference between the number of feature requests users express for more expensive apps and cheaper apps. With almost a quarter of the reviews providing feature requests, we argue that support in automatically retrieving feature requests from online reviews is needed.

III. TOOL SUPPORT

MARA is a prototype developed to mine for and retrieve feature requests from online reviews of mobile apps. The system is designed to (Figure 1): 1) retrieve all the reviews available for an app (*Review retrieval*), 2) mine the content of the reviews for identifying sentences or fragments of sentences expressing feature requests (*Feature requests mining*), 3) summarize such content (*Feature requests summarization*), and 4) present it in a user-friendly manner (*Feature requests visualization*). During the *review retrieval* phase, a web crawler extracts the page sources which make up the reviews of a given app (raw reviews) and parses their content. Of interest to this work is the actual content of reviews, but meta-data associated with each review is also collected for further analysis. Such meta-data includes the date the review was posted, the rating the user gave, the device associated to the review, the version of the app used by the user, and the title the user associated the review with. Further analysis could include a chronological evaluation of the evolution of feature requests for apps. Both the review's content and the review's meta-data are stored, the content being normalized to reduce the noise in the final results. Additionally, the reviews content is split into sentences. For that, we use LangPipe [8], a natural language processing tool which supports sentence splitting.

The *feature requests mining* phase takes as input the split review's content and mines for feature requests expressed by users. The feature request mining algorithm uses a set of linguistic rules defined for supporting the identification of sentences which refer to such requests. For example, feature requests are more prone to be expressed in sentences such as 'Adding an exit button would be great', which translates into a linguistic rule of the form 'Adding <request> would be <POSITIVE-ADJECTIVE>'. During the *feature requests summarization* phase, the system summarises the extracted feature requests according to a set of predefined rules. The goal of these rules is to rank the extracted requests based on their frequency and length; more frequent and lengthier feature requests would show up first in the summary. This is a necessity due to the large amount of feedback users provide. Lastly, during the *feature requests visualization* phase, the results of the summarization are displayed to the user. Through this paper, we will address the process of feature request mining and its evaluation.

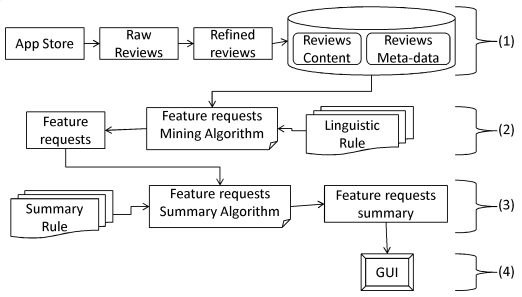

Fig. 1 – MARA prototype architecture

A. Feature Requests Mining

Based on the analysis of the manually identified feature requests in the random sample described in Section II, we explored the possibility of defining a language for expressing requests. First, we went through all the sentences labelled as feature requests and we associated each with a keyword which denotes the sentence as a request. We selected all those keywords which were associated with more than 3 sentences in order to avoid accidental associations of words with feature requests. We identified 24 such keywords (Table 1), which 80% of the sentences contained.

TABLE I. KEYWORDS FOR EXPRESSING FEATURE REQUESTS

add, *allow*, *complaint*, *could*, *hope*, *if only*, *improvement*, *instead of*, *lacks*, *look forward to*, *maybe*, *missing*, *must*, *needs*, *please*, *prefer*, *request*, *should*, *suggest*, *waiting for*, *want*, *will*, *wish*, *would*

Having identified such keywords, we filtered all those sentences containing at least one of these keywords and we went through them all in order to define the contexts (i.e. fragments of sentences) in which these keywords point to actual feature requests. Examples of such contexts are: "an exit button *would* be fantastic", "adding more icons *would* be great", "tips and math support *would* also be nice". Then we abstracted contexts into linguistic rules such as: "*(adding) <request> would (<ADV>) be <POSITIVE-ADJECTIVE>*". The word "adding" is optional, while ADV and POSITIVE-ADJECTIVE can be replaced by any adverb or positive adjective, respectively. Using all the contexts identified, we defined a set of 237 linguistic rules, some illustrative examples being depicted in Table II. Identifying a feature request translates into identifying contexts which match at least one linguistic rule. A limitation of this work relies in only considering reviews written in English. However, the approach can be applied to other languages by defining language specific linguistic rules. Also, sarcasm and irony are not specifically addressed by this work.

TABLE II. LINGUISTIC RULES FOR DEFINING FEATURE REQUESTS

Linguistic Rule	Example Context
<request> *would make it* <COMPARATIVE-ADJ>	"support for VTODO *would make it much cooler*"
(<SB>) (<ADV>) wish there was <request>	"*I just wish there was* the smiley editor ability"
<request> *should be* <COMPARATIVE-ADJ > *than* <existing-feature>	"the long press *should be shorter than* 0.25 seconds"
wish < request> *instead of* <existing feature>	"*Wish* the 2 add-ons were in a bundle pack *instead of* doing two transactions"
please include <-request>	"Next update *please include* a journaling feature with a keyword search"
could use (more) < request>	"*Could use more* icons"; "*could use* zoom and horizontal layouts"
add the ability to <request>	"*Add the ability to* create walls so they don't go off screen and to make cool mazes"
(the only thing) missing <request>	*"The only thing missing* is font customizations"
needs the ability to <request>	"*Needs the ability to* set custom wall paper"

B. Evaluation

In the context of feature request mining, we consider the following elements for the definition of performance metrics: a) true positives (TP) as the correctly returned feature requests, b) false positives (FP) as the returned results which are not actual feature requests, c) true negatives (TN) as the non feature requests not returned as results, d) false negatives (FN) as actual feature requests not returned as results. As metrics, we consider precision, recall, and Matthews correlation coefficient (MCC). *Precision* is the ratio between the returned results which are actual feature requests (TPs) and the total number of returned results (TPs + FPs). *Recall* is the ratio between the returned results which are actual feature requests (TPs) and the total number of feature requests in the input (TPs + FNs). Lastly, MCC is defined as (with MCC = 1 as a perfect predictor):

$$MCC = \frac{TPXTN - FPXFN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

For evaluation purposes, we downloaded the reviews of half of the available non-free apps on Google app store (on the 7th of February 2013) and we parsed them to retrieve their content and metadata. We chose these apps because of the large number of reviews they are associated with; the pool we stored for analysis comprised 136,998 reviews. We used these reviews as input for the feature requests mining algorithm (after having split their content into sentences) and we evaluated the results based on the metrics above. We randomly selected 3000 feature requests returned by the algorithm and a human coder analyzed the sample to check which were FPs. The task of the coder was to identify for each result analyzed the actual request the user expressed. Those results for which no such request could be identified were considered FPs. Overall, the precision calculated for the sample was P = 0.85. In terms of recall and MCC, we randomly selected one of the apps and choose its reviews as a sample for measuring the two metrics. We looked at a sample of 480 reviews, split into 778 sentences and the results are depicted in Table III.

TABLE III. RECALL AND MCC METRICS

Inputs	TP	FP	TN	FN	R	MCC
778	65	3	701	9	0.87	0.90

IV. Feature Request Analysis

We used LDA to identify topics among the feature requests obtained as result of our evaluation. We considered each feature request as being a document and we aimed to identify a set of topics described by a set of keywords which would be associated with the entire corpus of documents. To reduce the noise in the final results, we extracted both English stop words and the keywords identifying feature requests (see Table I) from the entire corpus. For better precision, we ran the LDA algorithm using different combinations of number of topics and keywords (Table IV). Additionally, identifying one topic for the entire corpus leads to associating it with the following keywords (game$^{(0.015)}$, app$^{(0.011)}$, use$^{(0.010)}$, update$^{(0.008)}$, levels$^{(0.007)}$, feature$^{(0.006)}$, work$^{(0.006)}$, better$^{(0.0057)}$. Overall, LDA identified several topics over the corpus used – (*game*$^{(0.130)}$), (*use*$^{(0.097)}$), (*update*$^{(0.076)}$), (*app*$^{(0.061)}$), (*feature*$^{(0.056)}$), (*support*$^{(0.048)}$), (*options*$^{(0.022)}$). *Game* is a recurring topic associated with keywords such as "update", "levels", "better"; users often asking for game apps to be frequently updated with more levels. As a topic, *App* is often associated with keywords such as "support", "way", "feature", "ability". On analysis of the reviews, it becomes clear that users use the words "way", "option", and "feature" to refer to the possibility of performing a specific action when using the app. Users ask for better support and customization options for the apps they use. *Update* stands out as a topic associated with keywords such as "levels", "better", "support", "stars".

TABLE IV. FEATURE REQUESTS TOPICS (COLUMN) ASSOCIATED WITH KEYWORDS (ROWS) AND THEIR PROBABILITIES

	1	2	3	4	5
1	(*game*$^{(0.011)}$)	(*app*$^{(0.021)}$), (*game*$^{(0.022)}$)	(*app*$^{(0.033)}$), (*game*$^{(0.035)}$), (*use*$^{(0.031)}$)	(*app*$^{(0.045)}$), (*game*$^{(0.045)}$), (*update*$^{(0.032)}$), (*use*$^{(0.041)}$)	(*game*$^{(0.070)}$), (*app*$^{(0.056)}$), (*use*$^{(0.043)}$), (*update*$^{(0.039)}$), (*way*$^{(0.017)}$)
2	(game$^{(0.011)}$, app$^{(0.011)}$)	(*app*$^{(0.023)}$, use$^{(0.020)}$), (*game*$^{(0.023)}$, update$^{(0.016)}$)	(*app*$^{(0.031)}$, work$^{(0.018)}$), (*game*$^{(0.034)}$, update$^{(0.023)}$), (*use*$^{(0.030)}$, better$^{(0.017)}$)	(*game*$^{(0.046)}$, levels$^{(0.025)}$), (*app*$^{(0.044)}$, great$^{(0.024)}$), (*update*$^{(0.031)}$, work$^{(0.024)}$), (*use*$^{(0.040)}$, better$^{(0.022)}$)	(*app*$^{(0.057)}$, use$^{(0.050)}$), (*update*$^{(0.038)}$, make$^{(0.037)}$), (*feature*$^{(0.030)}$, better$^{(0.028)}$), (*game*$^{(0.069)}$, levels$^{(0.032)}$), (*time*$^{(0.025)}$, way$^{(0.018)}$)
3	(game$^{(0.011)}$, app$^{(0.011)}$, use$^{(0.010)}$)	(*app*$^{(0.023)}$, use$^{(0.020)}$, make$^{(0.015)}$), (*game*$^{(0.023)}$, update$^{(0.016)}$, levels$^{(0.013)}$)	(*app*$^{(0.031)}$, make$^{(0.018)}$, nice$^{(0.016)}$), (*game*$^{(0.034)}$, update$^{(0.023)}$, levels$^{(0.020)}$), (*use*$^{(0.031)}$, get$^{(0.015)}$, support$^{(0.015)}$)	(*update*$^{(0.031)}$, work$^{(0.024)}$, support$^{(0.020)}$), (*use*$^{(0.035)}$, nice$^{(0.021)}$, screen$^{(0.014)}$), (*app*$^{(0.036)}$, get$^{(0.020)}$, time$^{(0.019)}$), (*game*$^{(0.046)}$, work$^{(0.024)}$, support$^{(0.020)}$)	(*feature*$^{(0.030)}$, nice$^{(0.026)}$, able$^{(0.017)}$), (*game*$^{(0.070)}$, levels$^{(0.033)}$, make$^{(0.037)}$), (*app*$^{(0.053)}$, update$^{(0.036)}$, work$^{(0.030)}$), (*use*$^{(0.052)}$, great$^{(0.030)}$, support$^{(0.026)}$), (get$^{(0.024)}$, *time*$^{(0.024)}$, way$^{(0.017)}$)
4	(game$^{(0.011)}$, app$^{(0.011)}$, use$^{(0.010)}$, update$^{(0.008)}$)	(*app*$^{(0.022)}$, use$^{(0.016)}$, great$^{(0.012)}$, nice$^{(0.011)}$), (*game*$^{(0.023)}$, update$^{(0.016)}$, make$^{(0.015)}$, levels$^{(0.013)}$)	(*app*$^{(0.034)}$, nice$^{(0.016)}$, time$^{(0.015)}$, phone$^{(0.013)}$), (*game*$^{(0.033)}$, update$^{(0.023)}$, levels$^{(0.020)}$, make$^{(0.022)}$), (*use*$^{(0.031)}$, great$^{(0.016)}$, get$^{(0.015)}$, support$^{(0.015)}$)	(*app*$^{(0.043)}$, get$^{(0.020)}$, work$^{(0.024)}$, time$^{(0.020)}$), (*update*$^{(0.032)}$, levels$^{(0.027)}$, better$^{(0.021)}$, support$^{(0.019)}$), (*game*$^{(0.046)}$, make$^{(0.030)}$, great$^{(0.024)}$, play$^{(0.020)}$), (*use*$^{(0.037)}$, nice$^{(0.019)}$, able$^{(0.014)}$, feature$^{(0.011)}$)	(*game*$^{(0.071)}$, make$^{(0.036)}$, better$^{(0.028)}$, play$^{(0.025)}$), (*levels*$^{(0.034)}$, get$^{(0.025)}$, new$^{(0.022)}$, options$^{(0.013)}$), (*feature*$^{(0.030)}$, nice$^{(0.026)}$, way$^{(0.018)}$, able$^{(0.017)}$), (*app*$^{(0.056)}$, update$^{(0.037)}$, support$^{(0.025)}$, stars$^{(0.022)}$), (*use*$^{(0.051)}$, work$^{(0.030)}$, time$^{(0.024)}$, phone$^{(0.022)}$)
5	(game$^{(0.011)}$, app$^{(0.011)}$, use$^{(0.010)}$, update$^{(0.008)}$, make$^{(0.007)}$)	(*app*$^{(0.023)}$, use$^{(0.020)}$, make$^{(0.015)}$, nice$^{(0.011)}$, support$^{(0.010)}$), (*game*$^{(0.023)}$, update$^{(0.016)}$, levels$^{(0.013)}$, great$^{(0.012)}$, work$^{(0.012)}$)	(*app*$^{(0.033)}$, nice$^{(0.016)}$, time$^{(0.015)}$, way$^{(0.011)}$, screen$^{(0.011)}$), (*game*$^{(0.035)}$, update$^{(0.023)}$, make$^{(0.022)}$, levels$^{(0.020)}$, better$^{(0.017)}$), (*use*$^{(0.029)}$, work$^{(0.017)}$, support$^{(0.015)}$, get$^{(0.015)}$, new$^{(0.013)}$)	(*app*$^{(0.037)}$, great$^{(0.023)}$, nice$^{(0.022)}$, support$^{(0.019)}$, way$^{(0.014)}$), (*update*$^{(0.031)}$, work$^{(0.023)}$, stars$^{(0.017)}$, phone$^{(0.017)}$, screen$^{(0.014)}$), (*use*$^{(0.041)}$, feature$^{(0.024)}$, better$^{(0.022)}$, time$^{(0.021)}$, options$^{(0.010)}$), (*game*$^{(0.057)}$, make$^{(0.029)}$, levels$^{(0.024)}$, get$^{(0.020)}$, play$^{(0.020)}$)	(*game*$^{(0.071)}$, make$^{(0.034)}$, play$^{(0.025)}$, great$^{(0.022)}$, good$^{(0.019)}$), (*work*$^{(0.027)}$, support$^{(0.023)}$, phone$^{(0.022)}$, screen$^{(0.018)}$, fix$^{(0.014)}$), (*levels*$^{(0.033)}$, time$^{(0.025)}$, get$^{(0.025)}$, new$^{(0.022)}$, waiting$^{(0.013)}$), (*use*$^{(0.051)}$, nice$^{(0.028)}$, better$^{(0.021)}$, able$^{(0.017)}$, music$^{(0.012)}$), (*app*$^{(0.056)}$, update$^{(0.039)}$, feature$^{(0.031)}$, stars$^{(0.022)}$, ability$^{(0.014)}$)

V. EASE OF USE

Through this work, we aim to support app developers in retrieving and making sense of the feature requests that users express through online reviews. We developed a system able to extract feature requests from online reviews of mobile apps and we used it to extract such requests from a sample of 136,998 online reviews. We then used the LDA model to identify topics that can be associated with these requests. We learned that most of the users' requests concern improved support for apps, more frequent updates, new levels for game apps, and more customization options.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Varsha Veerappa.

REFERENCES

[1] D. Sanjiv, M. Chen, "Yahoo! for Amazon: Sentiment extraction from small talk on the web", Management Science, 53(9), 1375-1388, 2007.

[2] J. A. Chevalier, D. Mayzlin, "The Effect of Word of Mouth on Sales: Online Book Reviews", Journal of Marketing Research, 43(3), 345-354, 2006.

[4] M. Harman, Y. Jia, Y. Zhang, "App store mining and analysis: MSR for app stores", MSR 2012, pp.108-111, 2012.

[5] I. J. M. Ruiz, M. Nagappan, B. Adams, A. E. Hassan, "Understanding Reuse in the Android Market", ICPC2012, 113-122.

[6] M. Hu, B. Liu, "Mining and summarizing customer reviews", KDD2004, 168-177, 2004.

[7] D. M. Blei, A. Y. Ng, M. I. Jordan, "Latent Dirichlet Allocation", Journal of Machine Learning Research, vol. 3, 2003.

[8] http://alias-i.com/lingpipe/index.html