EasyChair Preprint

№ 15527

Mining App-Specific Opinions in Mobile App Reviews Using Lexico-Syntactic Patterns

Phong Vu and Tung Nguyen

December 4, 2024

Mining app-specific opinions in mobile app reviews using lexico-syntactic patterns

***Abstract*—Mobile app markets are increasingly crowded and competitive. User opinions in app reviews are important for app developers to improve users' experience and satisfaction with their apps. However, current approaches for automated analysis of mobile app reviews (e.g. SURF, AR-Miner) have two problems: firstly, developers do not have the ability to define their own topic of interest to query relevant information from app reviews, as these approaches focus more on the common and established aspects of the app (e.g. GUI, connection, payments, etc); secondly, while some of them use linguistic patterns to classify the intentions of each opinion, they suffered from low recall due to the messy nature of review text (e.g. SURF has 22.5% recall for feature request classification). To address these problems, in this paper, we introduce MARP as a semi-automatic tool for developers to search and classify their opinions of interest using a Lexico-Syntactic Pattern approach. A Lexico-Syntactic Pattern is a sequence of tokens including syntactic lexicons (e.g. 'have' or 'please') and content fragments (e.g. OBJECT or ACTION) that together represent the syntax pattern of an intention. Our experiments show that MARP can extract complaints and requests more accurately than the baseline approach and is rated with comparable usefulness by app developers.**

***Index Terms*—phrase, template, app, review**

I. Introduction

Mobile app development is becoming more and more competitive with more than a million of available apps on popular app market such as Google Play. This number is still growing, leads to a strictly competitive environment for app development. Newcomers want to know how users think of the available apps to make better ones and the old players also need that information to improve their owns. One simple way to get that kind of information is to read user reviews on the rating system of app markets. However, in reality, it is not as easy as it sounds since each apps could have millions of review. Therefore, a research trend was formed to analyze app reviews, including classifications, information extraction by keywords, topics, specific text structures or just exploring the possibilities of user reviews.

Currently, many state-of-the-art tools[7, 9, 13, 32] have been introduced with novel and unique approaches. Those tools have excelled in determining the common sentiments and opinions from app reviews as a whole. For example, developers can easily use SURF or AR-miner to find out how the customers think of several aspects of the apps. Those aspects have been proven to be interesting and useful by actual developers [9]. However, while the covered topics are representative to many apps, they may not be able to address specific needs of a single app, such as when a developer need to know how customers think about their new or unique feature. Apps, much like any other type of software, constantly innovate. Each new app could introduce many special aspects that were not widely presented before. Hence, the common topics may not apply to them. We believe the developers' desire to understand what customers discuss/talk about a specific topic really exists and is crucial to app development. Let us demonstrate one example of such needs via a case-study of Magic Tile 3 below.

Magic Tiles 3 was developed by AMANOTES as a piano tapping game for mobile phone. The app has between 50 millions to 100 millions downloads on Google Play and generally get a positive feedback from its customer with average rating of 4.5 up to the date of January 2018. We have interviewed the developers of this app to ask them what are the user's opinions they want to learn about their app, and if there is any automated tool helping them to do that. Their answers were that they want to know what their users suggestions for the songs, what they think about their main competitor Piano Tiles 2 and what they think about the new Battle Mode. Normally, they would have to hire a team of customer service employees to read through all the comments and write a summary for the developers to update the app.

As we can see from this case, app developers wanted to find user opinions based on their intention (i.e. bug report, feature request, etc) about a few specific topics of interest.

With this requirement, we searched through the literature and found several state-of-the-art tools that may be able to help with their queries: AR-Miner, MARK, and SURF. However, none of them can point to the exact opinion without substantial human effort. AR-Miner can help finding informative reviews, but not just the ones belong to our topic about songs, Piano Tiles 2 and Battle Mode. SURF can find *suggestions, bug reports* amongst other intentions for 12 fixed general topics, however, none of those topics was asked by the developers. Lastly, MARK can find the keywords of queried topics, then search the reviews containing those keywords, but could not determine their intentions. Even though MARK comes close to answering the queries, human resource is still needed to infer what the reviews actually say about those topics.

From this observation, we believe that the current state of the art tools are not sufficient to give developers a report of what they want to learn from users about their various topics of interest. Therefore, in this paper, we propose MARP as a robust tool to help developers analyze specific topics on their own. It allows them to start with a set of words or phrases as a topic and an intention that they wish to learn about. After that, it automatically searches for text sequences that matched with such topic and intention. Furthermore, if the analysts want to

create, or to extend an intention, they can define such intention by adding linguistic patterns for it and let our tool find similar patterns in the text automatically. Our developer evaluations indicated that developers with specific topics in mind found our tool more useful than the state-of-the-art tools.

A linguistic pattern is a commonly used grammatical structure. For example, "[somebody] should [do] [something]" is a linguistic pattern used for suggestions or requests. For example, the app review *"You should add some Hindi songs"* matches the aforementioned pattern. In previous studies [9,10], linguistic patterns have been used as an effective approach to classify intention in text. However, those patterns were not easy to obtain, as they require manual effort. More importantly, users often use slightly different patterns to describe the same idea. This causes the Parse Tree approach to suffer from even the smallest changes in the tree structure of users' sentences, leading to a low recall rate in classification (e.g., recall rate for extracting feature requests is 22.5% [28]).

MARP has several advantages over the state-of-the-art tool in classifying intentions by patterns. First, rather than using ambiguous patterns like "[somebody] should [do] [something]", MARP uses Lexico-Syntatic ones.A Lexico-Syntactic pattern is a sequence of tokens including fixed syntactic lexicons (e.g. should or please) and content fragments (e.g. OBJECT or ACTION) that together represent the syntax pattern of an intention. For example, "[OBJECT] should [ACTION]" is a Lexico-Syntactic Pattern. In this pattern, [OBJECT] is the placeholder for an object/entity, [ACTION] is for an action. [OBJECT] and [ACTION] inferred from the word sequence by several English heuristics. MARP defines its Lexico-Syntatic Patterns as sequences of tokens, each could be a fixed word like 'should' or a placeholder. We have defined a similarity measurement for such linguistic patterns, thus, MARP can expand its set of patterns from data. In addition, we designed a near-matching algorithm to match these patterns to text. This algorithm is sequence-based, which does not require traversing parsed trees. Our experiments have shown that it is much more efficient and effective than the Parse Tree traversal method.

Overall, MARP proposes to solve the problem of finding specific opinions on reviews using topic keywords and Lexico-Syntactic Patterns. Our technical contribution includes the following:

- A complete framework to find opinion about specific topic using Lexico-Syntactic patterns and keywords.
- An overhaul of Lexico-Syntactic Pattern for flexible pattern matching, including a the definition of Expandable Lexico-Syntactic Pattern (ELSP) and a similarity measurement for such patterns.
- A semi-automated ELSP expansion approach.
- A Near-Matching algorithm for matching of ELSP on text.

Moreover, we have conducted two experiments to compare the sensitivity and effectiveness to a baseline approach. We have also asked developers of three apps to evaluate MARP 's usefulness comparing to SURF's.

The rest of this paper is organized as following: Section II discusses the motivation for our work. Section III describes the ELSP definition and modeling. Section IV describes the techniques used in MARP . Section V and Section VI evaluate our approach. Finally, we discuss the related work in Section VII, then threats to validity in Section VIII. Section IX concludes our paper and discusses future work.

II. Motivating Example

To build a system that could provide developers opinions with the flexibility of both topic and intention, we have tried different state-of-the-art tools and techniques. From them, we have concluded that intentions in reviews could be inferred by their linguistic patterns [10] such as the case of DECA. In DECA, the linguistic pattern would be hard-coded into a code snip-set that travels and checks the Parse Tree of a sentence to see if its features fit the pattern. However, in our experiments using the data described in Section V, the recall rate seems to be low (particularly in requests with the recall of 40.2%). Further observation showed that, even though the patterns themselves are correct, the variations in the way people write reviews made their sentence structures vastly different. For example: "fix the sync problem please" is not recognized as a problem discovery by DECA, but "please fix the sync problem" is even though these two sentences are the same to human reader in term of intention. Further investigation revealed that those two sentences have vastly different Parse Trees structures. With just one word "please" placed in a different location, the Parse Tree technique was not able to match a pattern for the former sentence. In another more complex example, "I would like to have ads removed please" is recognized as a request by DECA, while "I would like to remove ads please" is not. In this case, not only the word "ads" and "remove" switched their positions, but the latter also did not contain "have". This slight variant of the same sentence again prevented the Parse Tree technique to match the correct pattern.

In the two examples above, we have seen that the smallest difference in the way users write their review could lead to a drastic change in the structure of the Parse Tree, making matching by Parse Tree ineffective. Theoretically, this problem could be solved by adding more slightly changed linguistic patterns to fit with all the variants users can make. However, given the messy nature of reviews [32] and the flexibility of English, those variants could be too many to manually address.

Last but not least, our experiment showed that DECA takes a significant amount of time (Table II) to classify a large number of reviews. This was because of the computational power needed to traverse Parse Trees. If there are more patterns to compare, this processing time can take much longer. We believe that developers or analysts in charge of millions of reviews may need a faster solution.

Therefore, we need a new method that is not only flexible in classifying intention from reviews, but also is able to adapt to the way users write them. Moreover, the new method needs

to be reasonable in classifying speed. Therefore, we propose our Expandable Linguistic Model as a solution.

III. Expandable Lexico-Semantic Model

A. Expandable Lexico-Semantic Pattern (ELSP)

Let's examine the example "I would like to have ads removed please" . This sentence can be matched with this linguistic pattern: "[someone] would like to have [something] removed". In our observation, words such as "would", "like", "to", "have" and "removed" are fixed structural words (i.e. they serve as indicators of the structure of a pattern) that, when go together in a specific order (e.g. "have to" and "to have" describe different intentions), often indicate an intention, while the placeholders (i.e. [*someone*], [*something*]) are content to be filled in. Further observation from our dataset and Panichella *et al.* 's linguistic pattern set, we also discovered that these fixed words can include stop words, functional words, connectors (e.g. in, from, etc), intensifiers (e.g. more, most, too, few, etc), negations (e.g. not, never), question words (e.g. who, what, how, when, why). Moreover, domain/app specific words also play an important role in describing user intentions as well. For example, "boring", "repetitive", "additive" may help indicate intentions for complaints or praising a feature of *game* apps, but are unlikely to be a common occurrence in a *music player* app, while "crash", "fix", "add" usually give the same meaning across all mobile apps.

With these observations, we believe that the subsequences of fixed structural words play a crucial role in describing a linguistic pattern, and it is advisable that each sub-category or even each app/domain should have different common patterns based on how there users use the subsequences of fixed words to describe their ideas.

Furthermore, as they are subsequences of words, it is feasible to discover more similar subsequences from the app data starting from a set of initial patterns by comparing their set of sub-sequences using JACCARD similarity. Example of the matching result can be shown in figure 1 (we set max length of a subsequence at 2 for a simple calculation)
ϕ_i : Lexico-Syntactic Patterns i
ξ_i : A set of fixed word subsequences belong to ϕ_i

$$patternSim(\phi_1, \phi_2) = \frac{|\xi_1 \cap \xi_2|}{|\xi_1 \cup \xi_2|} \tag{1}$$

However, the newly discovered patterns are still susceptible to the difficulty of parsing trees in messy mobile app reviews. Therefore, instead of parsing the reviews, we use the same JACCARD similarity to match our pattern with review text. This way, even if users did not put their punctuations correctly, their expressions can still be matched, and slight variation of the way they write their sentences would not prevent the matcher from discovering them.

Another concern is that our mined sequences may not be grammatically correct, which would make the matched sequences incomprehensible, even meaningless (e.g. "[someone] have to" is not a complete phrase/sentence, and offers no meaning). In a Parse Tree approach, this was not a concern, as the tree would include the dependencies that make the matched sequence coherent. However, as discussed before, the Parse Tree approach is not flexible and slow. Therefore, to balance out between grammar coherency and pattern flexibility, we only mine patterns from sequences that are considered phrases or sentences by the Parse Tree technique. This means, before the mining start, we only need to run a Parser one time on the original data to filter out incomprehensible sequences. After that, the matching will be done on sequences, not the tree. This would provide a side benefit of significantly faster matching time despite of having to match more patterns (The expanded patterns can be larger than the original set) because there is no need for the matching algorithm to create and traverse the trees. This side benefit can also be vital if the analyst needs to analyze millions of reviews (experiment in Section V-D).

Finally, in review text, to the best of our knowledge, it is not yet feasible to correctly determine vague concepts such as [something], or [someone], or [feature], or [do] (as a vague description of a verb) that are complete and relevant to the idea users are trying to express in text. The Parse Tree approach would treat these placeholders as POS tags and annotations of the tree nodes and would recognize them when it traverses by. Therefore, to capture the content inside placeholders of a pattern, we create compositions of Part-of-Speech (POS) tags that we called Content Fragments. They are combinations of different Penn Treebank[22] POS tags and fixed words (e.g. connectors like to, from, or and) that would represent object compositions (e.g. "sync problem" is an object), or action compositions(e.g. "synced to music" is an action). Our approach is more heuristic-based since we do not try to analyze the tree for reasons stated above. A group of POS tags, if abides by the common English rules to be a verb group, noun group, will be considered as one by our approach. The rules are deducted by our team from several English grammar learning sources [1, 8]. Figure 1 and Figure 2 show two example of how we group POS tags together to create new compositions. Note that, in this solution, we do not try to capture dependencies (e.g. some independent parts of the Parse Tree that would qualify as the objectives of a statement), since it is not necessary as long as we can match the pattern to the text. Moreover, a parse tree might not be reliable enough to show their dependencies if users write the reviews without punctuation or spell the words incorrectly, which is often the case of mobile app reviews [33].

To formalize this new concept, we call the new linguistic patterns as: Expandable Lexico-Syntactic Patterns (ELSP) (Definition III-A). Note that this is not the same as Lexico-Semantic Patterns defined by Jacobs *et al.* [18], since we focus on the syntactic structures, not the semantic components.

Definition 1: ***Expandable Lexico-Syntactic Pattern (ELSP)***

Fig. 1: Example of converting text sequences to ELSP and comparing their similarity (sub-sequence max length = 2)

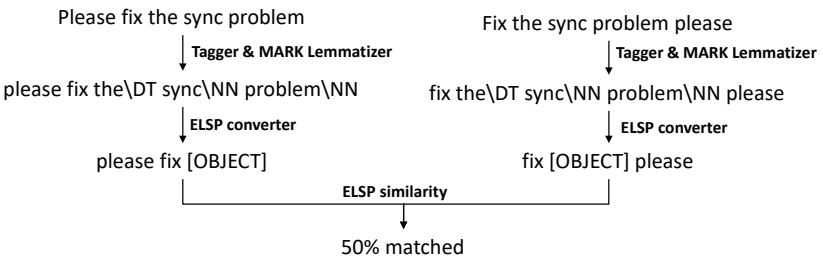
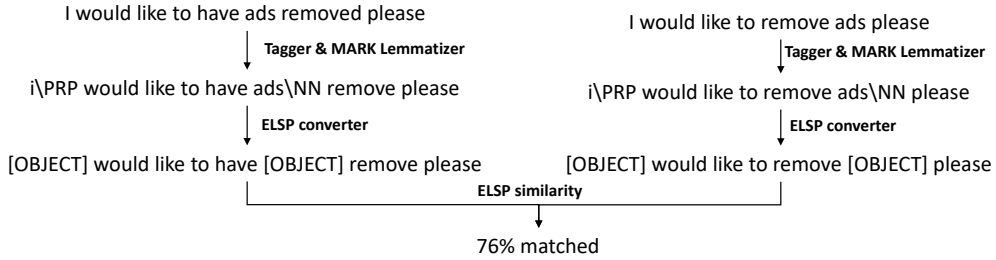


Fig. 2: Example of converting text sequences to ELSP and comparing their similarity (sub-sequence max length = 2)



is a sequence of Content Fragment Tokens and Lexico-Syntactic Tokens (LST), that is: [1] grammatically sufficient; [2] often indicates the same intentions to all text sequences it represents; and [3] has a similar intention as other ELSP that have similar subsequences of Lexico-Syntactic Tokens.

Definition 2: ***Content Fragment*** token is a composition of consecutive Part-Of-Speech (POS) tags and connector words (e.g. and, or, than, with, to) to describe an OBJECT (object composition), or an ACTION (action composition) .

Definition 3: ***Lexico-Syntactic Tokens (LST)*** are words that indicate the syntactic structure of a text sequence. The list of LST includes: connectors (e.g. and, or, than, with, to), intensifiers (e.g. more, much, less, many), WH (e.g. who, what, when, where), negations (e.g. not, never), and domain specific words (e.g. able, unable, freeze, force close, etc).

B. ELSP modeling

In this subsection, we describe step-by-step in detail how ELSP are obtained and utilized from review text, including: ELSP conversion, ELSP Expansion, and ELSP Near Matching.

1) ELSP conversion: To convert text sequences to ELSP, first, the module applies our composition heuristics to the sequences of POS tags and connector words (e.g. and, to, from, or, etc). This is to identify the Content Fragments and convert the whole text sequence into a sequence of tokens.

In the next step MARP uses Stanford Parser to extract templates that are annotated as phrases or sentence. This will create a database S_{struct} containing the ELSP templates that were determined as grammatically sufficient. In the ELSP matching step, S_{struct} is used to determine if a text sequence is grammatically sufficient by comparing its ELSP with this database. This step is OPTIONAL as it can be done on any dataset that has similar writings to mobile reviews. The more grammatically sound that dataset is, the more commonsense S_{struct} should be.

Lastly, we capture any sub-sequence of tokens from user data that also appeared in S_{struct}.

2) ELSP Expansion: In this section, we describe in detail how MARP expands a set of initial seeding pattern (S_i) into a bigger set of patterns (S_f) similar to them from data. The process has three steps:

First, we use Stanford Parser to identify the phrases (sentences or parts of sentence that have the annotation of phrase) in the chosen data. This means that the more carefully written data (with correct punctuations and correct spelling), the more correct the phrases would be identified.

Secondly, for each phrase or sentence, MARP identifies the LSTs and Content Fragments with the ELSP Conversion technique discussed above to produce their ELSP templates.

Lastly, MARP compare the ELSP templates from the second step with each of the patterns in S_i. If their similarity passes a certain threshold (chosen by user), they will be included in the final result set S_f as expanded patterns.

3) ELSP Near Matching: A "sentence" extracted from messy review text can sometime actually be composed of multiple actual sentences that were not punctuated correctly, which would result in two possible scenarios: Parse Tree does not represent the correct individual sentences and their structures; and multiple word sequences in that incorrect sentence would correctly represent our intention of interest. Moreover, a slight variation in the way user writes their opinion would make the matching using the tree annotations comparison less accurate and would negate possible matches.

Therefore, in our approach, we do not employ Parse Trees

```
function nearMatch
  Input: S_ELSP is a set of ELSP
         S_struct is a set of ELSP conforming a sufficient grammar structure
         W is an array of words representing a sentence
         ω threshold of ELSP similarity
  Output: S_F the set of sequences of words that matched
  S_F = ∅
  for (s = 0; s < W.length; s++)
    for (int e = W.length − 1; e > s; e−−)
      α = TextNormalizer.convert2ELSP(s,e,W)
      if(α ∈ S_struct)
        for each γ ∈ S_ELSP
          if (patternSim(α,γ) >= ω)
            S_F.add(getSequence(s,e,W))
            break;
  return S_F;
```

Fig. 3: Matching ELSP with the Near-Matching algorithm

in the process of matching with the ELSP set. Instead, we try to find the sequences of words that are *likely* to match the ELSP set. For a sequence to have the "likeliness" to match an ELSP, it has to conform two conditions: [1] It has to be grammatically sufficient (i.e. a typical phrase or a sentence, not an incomplete sequence such as "I think this feature is"); [2] Its ELSP form needs to be similar to the ELSP of interest.

In detail as shown in Algorithm figure 3, we need the input of S_{ELSP}, S_{struct}, W, and ω. In which, S_{ELSP} is the set of ELSP of an intention of interest obtained either manually or from the ELSP Expansion process; S_{struct} is a set of grammatically sufficient ELSP templates that was mined from the data in the preprocessing step; W is the sequence of words representing a sentence of interest from a mobile review; ω is a threshold of similarity for how much the ELSP form of a text sequence need to be similar to an ELSP in S_{ELSP}. In the algorithm, for each sequences within W (with the minimal length of 2 words), if its ELSP form belongs to S_{struct} and similar to at least one ELSP in S_{ELSP}, then we add that sequence to the matched set.

Note that, in this algorithm, we find all matched sequences, not just determining if the original sequence is matched or not. Therefore, the output would be all the sequences in W that conform our two conditions mentioned above.

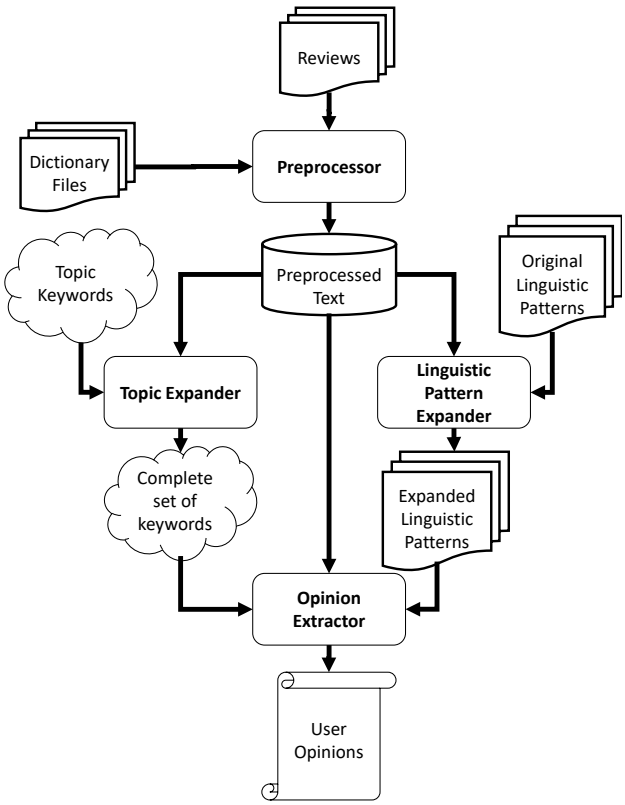
IV. Opinion Extraction with MARP

Definition 4: ***User Opinion*** of a topic is a fragment of a review that matches to a linguistic pattern of an intention and contains at least one keyword of that topic.

A. Overview of MARP

Opinion extraction for reviews has been discussed by Vu *et al.* and Panichella *et al.* in their previous publications. In their work, respectively, they define an opinion as a group of keywords or a sentence belong to a topic and a intention. Both of them have their strong sides: MARK is able to find customized topic that developers want to know about, while SURF can classify a sentence into one of the 12 major topics. However, SURF is limited to a few predefined topics and intentions, while MARK could not classify intention at all. In our evaluation in Setion VI, both could not answer the specific questions that were asked by app developers. In this

Fig. 4: Overview of MARP



section, we propose MARP as a robust approach to solve both problems, and to give developers what they desire to know about the opinions from their users. Furthermore, we hereby redefine User Opinion as in Definition 4.

As shown in Figure 4, MARP have two different use cases: Pattern Expansion and Opinion Extraction. The input for each task module can be modified outside of the framework to directly affect the results. Modifiable inputs includes: Dictionary files, Linguistic patterns, topic domain context-independent words. These inputs are included as artifacts for this paper, with the domain/functional words we used for reviews.[1]

The output for Pattern Expander is a list of expanded ELSPs, while the output of Opinion Extractor is list of reviews with highlighted opinion text and their intention. An example of this output can be found in Table III.

B. Preprocessing

App reviews are often messy [32]. Therefore, it is necessary to preprocess review text before moving further to other text mining phases. We tries to correct the commonly misspelled words in review using the mapper provided by MARK [34]. After that, MARP splits the review by sentences. In each sentence, it reduces words into their root form using a root and variation irregular dictionary inferred from Wordnet 3.0 [24] and from MARK's keywords. When creating this dictionary, we took out all irregular words (words that do not form their

[1] ARTIFACT: bit.ly/2UHEO6j

tenses using regular/established patterns, e.g. "begin, began, begun") from Wordnet 3.0 and app specific words obtained from MARK that did not appear in any dictionary (e.g. "noob" and "noobies", or "meme" and "memes"). This dictionary also indicates the correct POS tags of those words in their root form. For words that are regular and not app specific, we use MARK's custom stemmer to reduce them to their root forms and assign POS tag to them by Stanford POStagger [8]. During each steps, the process keeps track of the original positions of corrected/stemmed words in the original sentence. This is necessary in case users of MARP want to identify/highlight the analyzed text on the actual review later.

1) Topic Word Expansion: In Topic Word Expansion module, we use MARK framework proposed by Vu *et al.* to expand the given topic words into similar words using their vector representation[20]. The goal of this module is to expand the vocabulary of a certain set of topic keywords provided by users. This new vocabulary will then be used in the next module as a input for extraction of opinions about that topic.

2) ELSP extraction: Originally suggested by Vu *et al.* 's idea of extracting phrase template[33] for faster mapping of phrases, we expanded the concept of phrases into our concept of ELSP. Within Preprocessor module, if enabled, the phrases are originally extracted using Stanford phrasal extraction [8], before converted into an ELSP by creating Content Fragments. Users can also add more ELSPs for their domain of interest, enabling the capturing of heuristic patterns for specific domains. More detail is discussed in Section III.

Despite of the slow performance for ELSP extraction because of Parse Tree traversing, this process is only needed to be done once for any dataset or any group of similar datasets. The extracted patterns can be used on other datasets of the same domain, such as user reviews for mobile apps.

These patterns can be used as input for the optional Pattern Expansion module to find more patterns of a certain intention.

Once the user of MARP has the desired patterns, they can start the Near-Matching module, which takes input of user reviews, ELSP and topic keywords to find text clauses that fit such ELSPs and contain at least one topic keyword. More on how ELSP are matched using our Near-Matching technique is discussed in Section III-B3.

C. Opinion Extraction

After preprocessing, MARP requires user to clarify a topic by inputing a set of keywords belong to that topic. This step can both be done manually or automatically with MARK to give users freedom of choice over their topic. After choosing the topic, user should choose the intention they need for the analysis. MARP provides a default list of intentions including *request*, and *complaint*. User can also choose to use their own intention patterns to find a specific expression they are interested in and further expand them with ELSP Expanssion.

The process of extracting opinions is straight forward: MARP simply finds all the sequences of words that match with the provided ELSP patterns, then filter them by possession of keywords in the topic.

V. Comparing performance with state-of-the-art methods

In this section, we compare MARP with DECA and SURF as a baseline to demonstrate the sensitivity, and effectiveness. Firstly, we describe the datasets we collected and labeled. Secondly, we compare MARP with DECA for classification power of two intentions. Lastly, we compare the running time of two approaches and discuss how it is relevant to analytics.

A. Truth-sets for comparison

1) SURF truth-set: In the evaluations, we will compare with DECA for classification power as SURF is currently the state-of-the-art tool and their authors have confirmed with us that the linguistic classification module it uses is actually DECA. Therefore, we choose to use their dataset of 1,390 sentences to evaluate[2]. However, upon inspecting the dataset, we have found some inconsistencies that could affect the classsification results of both tools. For example, sentence "Please fix the bugs!!"(ID: 16378) is labeled as Problem Discovery but "Please fix this ASAP."(ID: 3820) is labeled as Information Giving. Further inspection found more similar inconsistencies (12% inconsistency). We believe this may affect the results from DECA. Therefore, we have decided to re-label this dataset.

We also have decided to rename the label of "Problem Discovery" to "Complaint", and "Feature Request" to "Request". This is because we believe user reviews present the perspective of users, not of developers, thus, anything that annoyed users would be seen as a problem [21]. Moreover, the requests from users can also be about other aspects of the app, not just features (e.g. *content, rewards, attention*, etc). Furthermore, the linguistic patterns provided by the authors of DECA also focused on general requests and complaints more than just problem discovery nor feature request. Therefore, if not classified by the more general concept of complaints and request, it may not be correctly classified by both tools. Lastly, Information Giving label is largely vague, as any informative piece of text can be labeled as Information Giving, even if it is a Request or Complaint. Other labels remained the same.

In the re-labelling process, we have 3 authors sitting together to discuss on each sentence's intention, then vote. We count a vote of any less than three members is a disagreement. The inter-rater agreement is 0.96 (almost perfect agreement), measured by Cohens kappa coefficient [31]. We only had different opinions over vague meaning sentences such as "My only problem is not being able to stop". In this sentence, the user could refer to either excitement over the app or a bug that prevent stopping the app. Two authors agreed that this sentence describes a complaint, and one author did not. In many other cases, we identify a sentence to be both a complaint and a request, such as the case of "Good app but wont let me get wifi please make it on a tablet". The previous label for this one was a problem discovery. In the end, the truth-set contains

[2]www.ifi.uzh.ch/en/seal/people/panichella/tools/SURF.html

Fig. 5: Perfomance (f-score) of MARP on classifying Complaint intentions two truth-sets by different thresholds of ELSP similarity

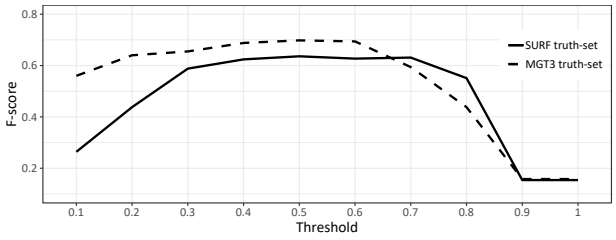
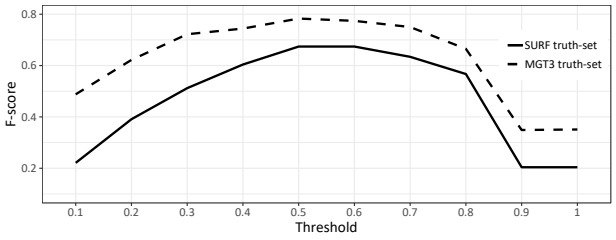


Fig. 6: Perfomance (f-score) of MARP on classifying Request intentions two truth-sets by different thresholds of ELSP similarity



558 complaints, 316 requests, in which 104 are both complaint and request. There are 620 sentences were labeled as neither.

After re-labelling this dataset, DECA seems to perform better than before, with higher precision and recall rates, ultimately leading to a slightly higher F-score (F-score of 0.69 comparing to 0.68 before for complaints and 0.38 comparing to 0.36 before for requests). We have concluded that our relabeling process has improved the performance of the dataset.

2) MGT3 truth-set: SURF truth-set has only sentences that were extracted by their authors. This may not be representative of what the actual data may look like. The real data should be full reviews, not sentences, and are often very messy [32]. While SURF is a good baseline comparision with previous tool, we still need another truth-set for the real world application. Therefore, we have contacted developers from AMANOTES to request a full set of reviews from Magic Tiles 3 (MGT3) for the whole year of 2017. We repeated the same labeling process one each review and reached an inter-rater agreement of 0.94 (Almost perfect agreement). Overall, the number of labeled reviews was 12,343 reviews, in which, 895 contains complaints, 378 contains requests, 82 contains both.

B. Similarity Threshold Evaluation

As an approach based on similarity measures, we need to test MARP 's sensitivities with different thresholds to determine which threshold is suitable for classification. Therefore, for this experiment, we apply different thresholds with an increment of 0.1 each step for both Complaint and Request intentions on both SURF truth-set and MGT3 truth-set. While we do aware that expanding patterns can be done multiple times, with each expansion's result being the input for the next expansion, resulting in a larger set of ELSP, we do not consider this case. This is because in a normal working scenario, we expect developers only need to expand their linguistic patterns into ELSP from their data with just one step for convenient.

The average F-score for each threshold are shown in Figure 5 and Figure 6. These results suggest that the threshold *0.5* seems to yield highest F-score for our truth-sets. Therefore, we used this threshold for all other experiments.

C. RQ1: How does MARP perform comparing to a similar state-of-the-art tool?

To evaluate sensitivity and effectiveness of our tool, we designed our tests to answer the research question: How does MARP perform comparing to DECA[10] in classification power with Expanded Linguistic Patterns?

In this experiment, we used the 65 Feature Request and Problem Discovery linguistic patterns from DECA as input for MARP to recognize Complaints and Requests. MARP expanded this patterns set based on data one time before classification. In this evaluation, we calculated recall, precision and F-score for both tools on SURF truth-set and MGT3 truth-set. There was no human intervention in this entire test.

The results are shown in table I. Overall, MARP seems to have better results than DECA in all aspects. This means that our ELSP approach captured more patterns than the old approach, and matched with more correct results. One interesting note is that given the differences in nature of the two truth sets, we can see DECA gives different results for each (i.e. in SURF truth-set, DECA works better with Complaints, while in MGT3 truth-set, it works better with Requests). This is expected, as DECA uses one linguistic pattern sets for both data, and given that users could express themselves differently on each dataset, their patterns may not perform similarly. This confirms our initial hypothesis that manually labeled linguistic patterns will suffer from various ways of user's expression. Meanwhile, MARP works consistently on both truth sets, as it relies on the data to expand its patterns.

TABLE I: Comparison between ALPACA and DECA over classification of Requests and Complaints

		Complaint		**Request**	
		ALPACA	DECA	ALPACA	DECA
SURF truth-set	Precision	73.5%	74.0%	76.1%	36.1%
	Recall	66.4%	63.8%	80.7%	40.2%
	F-score	**69.8**%	68.5%	**78.3**%	38.0%
MGT3 truth-set	Precision	68.9%	51.6%	62.4%	52.9%
	Recall	59.0%	21.1%	73.3%	53.8%
	F-score	**63.6**%	30.0%	**67.4**%	53.3%

D. RQ2: How efficient is MARP comparing to a similar state-of-the-art tool?

As we analyzed in Section II, DECA uses Parse Trees to match a pattern to a sentence, which could take a long time to run for a big dataset. For requirement analytics who need to work with millions of reviews to find out what user opinions are, that approach may not be the most efficient. That was one of the main reasons for MARP to approach it from the sequence matching perspective.

However, the authors of this paper are aware that if the processing time of DECA is not significantly long, then it may not be a problem at all. Therefore, in this experiment, we compare DECA and MARP running time on several datasets to see if both tools are reasonable in term of processing time.

We used 5 different datasets on this experience: SURF truth-set, MGT3 truth-set, Tankraid, Face Dance, and MARK's dataset. Tankraid (1648 reviews) and Face Dance (5094 reviews) are reviews provided by their respective developers for the entire year of 2017. MARK's dataset is the dataset of 3 millions reviews from VU *et al.* .

In this experiment, we use the same computer (Dell OptiPlex 7050 3.6GHz Core i7 16GB RAM with 5200rpm HDD) to run each dataset on both tools. To make sure there is no bias, we count both the preprocessing time and the classification time of our tool, since DECA also runs from preprocessing to classification. Please note that after preprocessing one time, our tool only need to run the classification step for any intention, which is significantly faster than running back from the beginning. Table II shows the results of our experiment.

On average, MARP can process about 2821 reviews per minute, while DECA can work 127 reviews per minute. This translates to MARP running with an average of **22.2 times faster** than DECA on almost all datasets. Regretfully, to our best effort, we could not finish our experiment with MARK dataset on DECA as it was taking an unreasonably long time and we could not be sure if there was problem occurred. Moreover, if the average reviews processed per minute holds true for this dataset, it could take around 6.5 days for DECA to finish. We believe a faster computer with an SSD can speed up this process. However, in real world applications, analytics may not have a fast enough computer to process millions of reviews with this approach in a reasonable amount of time.

In our conclusion, our new approach of matching ELSPs over text sequences gives a better, more consistent result (table I) in a significantly shorter time than the old approach of traversing through Parse Trees.

VI. Evaluation of usefulness from app developers

SURF has done an excellent job of evaluating its usefulness with developers of real apps. Even though MARP is addressing a different problem (finding opinions based on user-defined topics and intentions), it still makes sense that we need to evaluate if actual developers think this is useful. Moreover, since both tools approach finding opinions differently, we need to compare their approaches from app developers' view to see which one is preferable.

A. Data collection

Over the month of December 2017, We had sent out an participation invitation email to the app developers of 250 selected apps from Google Play. Our selection criteria for this evaluation consists of new apps in 2017 with less than 50k reviews (according to AppBrain [2]) as they are usually new and small apps with a small team. We believe it would be cost ineffective for them to hire a professional team of customer support employees or specialized analysts, thus, they would benefit the most from opinion mining tools. Our invitation email explains this research and how it can help them to understand their customers better through app reviews. In our original plan, to participate in our experiment, they would need to: [1] write to us of what they want to know from their reviews; [2] send use the reviews they have from the app stores for the time duration they want to investigate; [3] receive two reports from two anonymous tools (SURF and MARP); [4] complete a survey about the usefulness of each.

However, to our best effort in contacting the developers, only 3 app developers were interested and sent their data(Face Dance Challenge - 5,094 reviews; Tankraid - 1,648 reviews; Magic Tiles 3 - 12.343 reviews) along with the kind of user opinions they want to know. Therefore, instead of having them to fill in a survey, we designed two one-on-one interviews with a questionnaire for each of them to understand more about their needs and their evaluation after we sent them the reports. The first interview was over email, which had only one questions: what do you want to know from user reviews? We sent follow up emails to clarify that they understood the question to identify the topics and intentions that they wanted. The second interview happened after we sent them two reports from two tools. This interview is intended to evaluate both results and identify their preference. To eliminate bias, we packed the results of both tools in two similar excel files and did not disclose to them which tool we used. MARP also highlighted the review fragments that matched with the opinions, while SURF highlighted the sentences. In our questionnaire for evaluating the reports, we never mentioned which one was from our tool until after the interview. Our questionnaire and the reports are included in the artifact.

We also agreed to not share the personal identity of the developers who participate in our interviews, and only refer to them as representative of their companies/apps.

B. RQ3: Do developers think MARP is useful?

Overall, all three developers thought MARP was useful and helped them find what they wanted. Followings are individual reports from each developer:

1) Magic Tiles 3: Magic Tiles 3 is a piano tapping game developed by AMANOTE for both Android and iPhone. The app has between 50 millions to 100 millions downloads on Google Play alone.

During our email interview, we have learned that the developer wanted to know about three things at that point: how do user receive the music and songs, and do they have any suggestion? What do they report about the new Battle Mode? and do they compare Magic Tiles 3 to their main competitor, Piano Tiles 2? After more discussion, we also realized that they want to know about mostly feature requests and bug reports to improve their app.

With this interview, we prepared three topics in the report: songs and music, game mode, and piano tiles 2. We used MARP to find the keywords for those topics and ran them on their reviews, the whole process took about 15 minutes

TABLE II: Time required for classification using MARP and DECA on different datasets (in minutes and hours)

	SURF 1,390 reviews	**Tankraid** 1,648 reviews	**Face Dance** 5,094 reviews	**MGT3** 12,343 reviews	**MARK dataset** 3 millions reviews
ALPACA	31s	32s	2m 1s	4m 32s	18h 15m
DECA	13m	14m	41m	1h 18m	**>2d (not finished)**

including finding the keywords (The example report can be found in Table III). With the same amount of text on the same computer, SURF took 93 minutes in total.

The developers had responded that they liked both reports. However, there are queries about the app that they wanted but could not get from SURF reports, but got from our reports. SURF gave them the information they think is useful, while MARP gave them the information they asked for. In other word, both are useful to the developers.

TABLE III: Example report for Magic Tiles 3

Review	**Intention**	**Topic**
This game is cool but you should **add more game modes**	Request	Game mode
Please add battle with friend facebook .. Dnt battle random	Request	Game mode
Can u **add some hindi music** in it. It will be awesome	Request	Song and music
This is really good but it **needs some popular songs**	Request	Song and music
Song does not sync with my Samsung s8+. When fixed, will re-rate.	complaint	Song and music
Touching tiles is not synced to music! It's just background music! This **is a FAKE, KNOCKOFF of Piano Tiles 2!**	complaint	Piano Tiles 2

2) Face Dance Challenge: Face Dance Challenge is a relatively new app surfaced in the third quarter of 2017. With the unique idea of adjusting face to music and emojis, the app has become viral on social media and soon became one of the top ranking apps late year 2017.

At first, the developers simply wanted to know what users want to upgrade in the game (i.e. feature requests), therefore, we ran MARP without topic on their reviews. The results were various in topic, and we had received positive feedback from them, including a more specific topic about the "social" aspect of the game. Combining these feedbacks, in the final report, we decided to choose three topics to explore: Face detection and camera, Social, and in-app-purchase.

Including exploring the topics, MARP took 15 minutes to run through 5,095 English reviews to produce the report. SURF took 52 minutes for the same task. Examples of our report is shown is table IV.

After reading and comparing the two reports, the developers of Face Dance Challenge have given us the following feedback: Both reports are useful to what they do, however, they would prefer to read our report because of the detailed summary of opinion in the case they have to read more reviews. When revealed that our report can offer more topics they can define, they immediately asked for the topic "upgrade", which was not covered by both tools in the reports. After more clarification, we had concluded that this topic is simply "feature request", and MARP can simply list all of which without any topic keyword.

TABLE IV: Example report of Face Dance Challenge

Review	**Intention**	**Topic**
Can it **be played by two people in one camera**? Cause **I would like to try this with my girlfriend**	Request	Face Detection and Camera
This is so cool.**But please make the controls of the face easier**.But this is still fun so much..	Request	Face Detection and Camera
Try to just stare and do nothing while the game is on, you'll **still get perfect scores without moving your face**. -the game's face recognition is not accurate.	complaint	Face Detection and Camera
It's fun and all but when you try to save your video onto your gallery and then you close the app, the video wont save and will just say error occurred. **I tried sharing it to facebook or something or sending the video to my friends and the file will say it's corrupted.**	complaint	Social

The developers also made a suggestion, which is to list the results by versions. For now, MARP and SURF both do not have this functionality, as we plan to add it into MARP later.

3) Tank Raid Online: Tank Raid Online is an online multiplayer game developed by Wolffun Game in which people can play as tanks. The additive nature of the game resulted in 5 millions downloads by the end of 2017.

When contacted, its developers only showed interest for anything that is bug (bug report). Therefore, we ran MARP specifically for complaint without any topic and reported it back to them. They have verified to us that the founded bugs were consistent with what they had found by themselves. To further elaborate the results, we had extracted the topic from this finding and classified them into three main topics: Control, Battle and Match Making, In-app-purchase.

Both tool took the same amount of time to generate the reports, which was about 15 minutes. The example report by MARP can be seen in table V.

After carefully comparing two reports, developers of Tank Raid thought both are useful to them. However, they liked the report with highlighted opinions (MARP) better and expressed that they would even pay to use a tool like that with the ability to choose topics of their interest. In conclusion, MARP is necessary to them.

VII. Related Work

There is a number of empirical and exploratory studies on the importance of app's reviews in app development process.

TABLE V: Example report of Tank Raid Online

Review	Intention	Topic
You should add a pause button	Request	Control
Need a fixated joystick	Request	Control
I've noticed a glitch when you die,when you are respawning you are still able to shoot someone and thus kill them.When **you hold down the shoot button and you die you can continue shooting**	complaint	Control
Please add tower mode....i will give 5 star for it....please...	Request	Battle and Match Making
Omgeee!! Good online tank game Nice game But it **should have a group or team battle And a guild for guild battles**...	Request	Battle and Match Making
In **the middle of battle, the screen loading and back to the home menu without reason.** I lost all my money in pay battle	complaint	Battle and Match Making

In [30], Vasa *et al.* made an exploratory study about how users input their reviews on app stores and what could affect the way they write reviews. Later, Hoon *et al.* [15] analyzed nearly 8 millions reviews on Apple AppStore to discover several statistical characteristics to suggest developers constantly watching for the changes in user's expectations to adapt their apps. Again on Apple App Store, an emprical study about user's feedback was made by Pagano *et al.* [26]. Similarly, Khalid *et al.* suggest that there are at least 12 types of complaints about iOS apps [19]. They explored various aspects that influent user reviews such as time of release, topics and several properties including quality and constructiveness to understand their impacts on apps. Palomba *et al.* had conducted a study [27] of whether user reviews really are taken into account by developers in app development. Bailey *et al.* [3] investigated in the feedback loops of iOS reviews and their behaviors.

Other than reviews, price and rating of apps can also affect how people provide their feedbacks, as suggested in [17] by Iacob et al. Meanwhile, Bavota et al.[4] studied the relationship between API changes and their faulty level with app ratings. Recently, Martin et al. [23] reported the sampling bias researchers might encounter when mining information from app reviews.

There were several works focusing on mining useful information from user's reviews, such as one from Chandy *et al.* [6] who proposed a classification model for spamming reviews on Apple AppStore using a simple latent model. Another team, Carreno *et al.* [12] extract changes or additional requirements for new versions automatically using information retrieval techniques. Guzman *et al.* [14]. extracts features from app reviews in form of collocations and summarizes them with Latent Dirichlet Allocation (LDA)[5] and their sentiment. Scalabrino *et al.* developed a CLAP[29] to help categorizing reviews by their information.

Some other works resulted in complete tool set or prototypes such as Wiscom [11] or MARA[16], or AR-Miner [7]. Chen et al. propose a computational framework to extract and rank informative reviews at sentence level. They adopt the semi-supervised algorithm Expectation Maximization for Naive Bayes (EMNB)[25] to classify between informative and non-informative reviews. To rank the reviews, they use a ranking schema based on the meta-data of reviews and suggest the most informative ones. Gao *et al.* also provided INFAR[13] to analyze app reviews from multiple analysis dimensions. Later on, MARK [34] proposed a keyword based approach to discovering topics and trends using their semantic meaning.

The most closely related work to MARP would be SURF from Panichella *et al.* This work helps developers find both intention and topic from reviews. However, their underlying method for SURF has some severe limitations: slow speed of intention classification and inability to expand topic without modifying the tool. Moreover, since the intention classification part was based on DECA[28], the classification power is limited to the cost of labeling linguistic patterns by human. Our work addresses all of these problem by introducing a new definition of Expandable Lexico-Semantic Pattern that can be expanded automatically and be matched without the cost of traveling Parse Trees. Moreover, we use MARK approach to topic from keywords to let user define their topic using keywords. In other words, MARP framework provides an unbounded, flexible, and robust environment for opinions discovering on user reviews.

VIII. Threats to Validity

The dataset is a threat to the validity of our evaluation, as our data may not be representative enough to draw conclusion over millions of reviews and apps. However, we have minimized this threat by providing two different truth-sets with different characteristics: one is from a previous publication, and one is the unaltered review set of an app for an entire year. We also offer the truth-sets as an artifact of this paper for any interested researcher to replicate our experiments.

IX. Conclusion

In this paper, we have discussed the need of app developers to find user opinions about specific topics and intentions of interest. Subsequently, we introduced MARP as a tool for solving this problem. We also proposed a novel definition of the new Expandable Lexico-Syntactic Patterns, along its characteristics of being able to represent intentions and to compute a similarity measurement to each other.

Finally, we have evaluated our approach on multiple aspects: Evaluation of similarity threshold for ELSPs; comparing to DECA in term of sensitivity, effectiveness, and efficiency; and conducting three case studies with real app developers to compare usefulness to SURF. Our evaluation has shown that MARP is not only faster, more sensitive than previous state-of-the-art tools but also is useful to developers.

For future work, we are planning to add more default intention patterns to the tool, and to explore the relationship between order and similarity of LSTs inside a pattern.

REFERENCES

1. *English grammar*, dictionary.cambridge.org (2018).
2. AppBrain, *Android statistics*, www.appbrain.com (2016).
3. Kendall Bailey, Meiyappan Nagappan, and Danny Dig, *Examining user-developer feedback loops in the ios app store*, Proceedings of the 52nd hawaii international conference on system sciences, 2019.
4. G. Bavota, M. Linares-Vasquez, C.E. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, *The impact of api change- and fault-proneness on the user ratings of android apps*, SE (2015).
5. David M Blei, Andrew Y Ng, and Michael I Jordan, *Latent dirichlet allocation*, Journal of machine Learning research **3** (2003), no. Jan, 993–1022.
6. R. Chandy and H Gu, *Identifying spam in the ios app store*, 2012.
7. N. Chen, J. Lin, S. CH. Hoi, X. Xiao, and B. Zhang, *Ar-miner: mining informative reviews for developers from mobile app marketplace*, Icse, 2014.
8. M.-C. De Marneffe, B. MacCartney, C. D Manning, et al., *Generating typed dependency parses from phrase structure parses*, Lrec, 2006.
9. A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, *What would users change in my app? summarizing app reviews for recommending software changes*, Fse, 2016.
10. A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. Gall, *Deca: development emails content analyzer*, Icse, 2016.
11. B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, *Why people hate your app: Making sense of user feedback in a mobile app store*, 2013.
12. L.V. Galvis C. and K. Winbladh, *Analysis of user comments: An approach for software requirements evolution*, Icse, 2013.
13. Cuiyun Gao, Jichuan Zeng, David Lo, Chin-Yew Lin, Michael R Lyu, and Irwin King, *Infar: insight extraction from app reviews*, Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering, 2018, pp. 904–907.
14. E. Guzman and W. Maalej, *How do users like this feature? a fine grained sentiment analysis of app reviews*, Re, 2014.
15. L. Hoon, R. Vasa, J. Schneider, and J. Grundy, *An analysis of the mobile app review landscape: Trends and implications*, 2013.
16. C. Iacob and R. Harrison, *Retrieving and analyzing mobile apps feature requests from online reviews*, Msr, 2013.
17. Claudia Iacob and Rachel Harrison, *Retrieving and analyzing mobile apps feature requests from online reviews*, Proceedings of the 10th working conference on mining software repositories, 2013, pp. 41–44.
18. Simone Scalabrino, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta, *Listening to the crowd for the release planning of mobile apps*, IEEE Transactions on Software Engineering (2017).
19. Paul S Jacobs, George R Krupka, and Lisa F Rau, *Lexico-semantic pattern matching as a companion to parsing in text understanding*, Speech and natural language: Proceedings of a workshop held at pacific grove, california, february 19-22, 1991, 1991.
20. Hammad Khalid, *On identifying user complaints of ios apps*, Proceedings of the 2013 international conference on software engineering, 2013, pp. 1474–1476.
21. Tomas M., Kai C., Greg C., and Jeffrey D., *Efficient estimation of word representations in vector space*, CoRR (2013).
22. Walid Maalej and Hadeer Nabil, *Bug report, feature request, or simply praise? on automatically classifying app reviews*, 2015 ieee 23rd international requirements engineering conference (re), 2015, pp. 116–125.
23. Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz, *Building a large annotated corpus of english: The penn treebank* (1993).
24. William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang, *The app sampling problem for app store mining* (2015).
25. George Miller, *Wordnet: An electronic lexical database*, MIT press, 1998.
26. Kamal Nigam, John Lafferty, and Andrew McCallum, *Using maximum entropy for text classification*, Ijcai-99 workshop on machine learning for information filtering, 1999, pp. 61–67.
27. D. Pagano and W. Maalej, *User feedback in the appstore: An empirical study*, Re, 2013.
28. Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia, *Crowdsourcing user reviews to support the evolution of mobile apps*, Journal of Systems and Software **137** (2018), 143–162.
29. S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, *How can i improve my app? classifying user reviews for software maintenance and evolution*, Icsme, 2015.
30. R. Vasa, L. Hoon, K. Mouzakis, and A. Noguchi, *A preliminary analysis of mobile app user reviews*, Ozchi, 2012.
31. Anthony J Viera, Joanne M Garrett, et al., *Understanding interobserver agreement: the kappa statistic*, Fam med **37** (2005), no. 5, 360–363.
32. P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, *Mining user opinions in mobile app reviews: A keyword-based approach (t)*, Ase, 2015.
33. P. M. Vu, H. V. Pham, T. T. Nguyen, et al., *Phrase-based extraction of user opinions in mobile app reviews*, Ase, 2016.
34. P. M. Vu, H. V. Pham, T. T. Nguyen, and T. T. Nguyen, *Tool support for analyzing mobile app reviews*, Ase, 2015.