Mining User Opinions in Mobile App Reviews with Linguistic Patterns

Phong Minh Vu, Tam The Nguyen, Tung Thanh Nguyen
Auburn University
lenniel@auburn.com,tam@auburn.com,tung@auburn.edu

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 are not effective to extract app-specific bug reports and feature requests. In this paper, we propose ALPACA , a novel approach for that problem based on linguistic patterns. A linguistic pattern is a sequence of tokens including both functional words (e.g. 'for' or 'to') and placeholders of compositions (e.g., verb and noun phrases). In ALPACA , we define a large collection of linguistic patterns for bug reports and feature requests. Then, ALPACA expands this collection with similar linguistic patterns found in data. Finally, ALPACA extracts words sequences in app reviews that match those patterns and classifies those sequences based on their intents and topics. Experiments show that ALPACA can extract bug reports and feature requests more accurately than baseline approaches which are rated with higher usefulness by app developers.

ACM Reference Format:
Phong Minh Vu, Tam The Nguyen, Tung Thanh Nguyen. 2018. Mining User Opinions in Mobile App Reviews with Linguistic Patterns. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Million of mobile apps are currently available on popular app markets like Google Play and Apple AppStore, making mobile app development more and more competitive. To improve users' experience and satisfaction with their apps, app developers need to know users' complaints of existing issues (e.g., bug reports) or suggestions for improvement (e.g., feature requests). User reviews of mobile apps on app markets often contain such useful opinions. However, popular apps often receives thousands to millions user reviews, thus, manually reading and extracting useful user opinions from those reviews are time-consuming and ineffective.

To support app developers in mining user opinions from app reviews, researchers have proposed several approaches [7, 10, 26], which can infer what topics discussed in reviews, classify reviews into different kinds of intentions (e.g., bug reports or feature requests), search for reviews on a specific topics, or summarize reviews into general topics (e.g., GUI, network...). However, those approaches still have limitations and could not fully address app developer's needs, such as searching or extracting bug reports on an app-specific feature.

Let us demonstrate such limitations via a real world case. 'Magic Tiles 3' is a piano tapping game for mobile phone developed by AMANOTES. By January 2018, the app has between 50 to 100 million downloads on Google Play and generally gets positive feedbacks from its users with an average rating of 4.5. The app's developers tell us that they are looking for user's opinions on three topics. First, what new songs users want to add to the app's song library? Second, how do users think about the new feature called 'Battle Mode'. And third, how users compare 'Magic Tiles 3' to its main competitor 'Piano Tiles 2'.

As we can see from this example, app developers are interested in user opinions about *specific features* of the app (e.g., 'song library', 'Battle mode', comparison to the competitor 'Piano Title 2'). Useful opinions should have clear intentions, such as suggesting improvements, reporting problems, or giving information. Normally, AMANOTES would hire a team of customer service employees to read through all the comments and summarize such users' opinions in a report which is used to guide the development of the next version. They were not aware of any tool available for such tasks.

To support those app developers with this problem, we studied the literature and found three state-of-the-art tools that may be able to help with their queries: AR-Miner [8], MARK [26], and SURF [10]. However, none of those tools can automatically extract the relevant opinions from raw text reviews without substantial human effort. For example, AR-Miner can determine if a review is informative or not, but cannot determine whether the review content is relevant to the specific topics of developers' interest. SURF can find reviews of specific intentions (e.g. suggestions or bug reports), but it summarizes them into 12 general topics (e.g., GUI, network). It does not allow developers to specify their own topics of interest. In contrast, MARK allows developers to specify their interests via keywords and finds reviews containing those keywords, but developers need to manually analyze, extract, and summarize useful opinions in those reviews themselves.

This example suggests that current state-of-the-art tools are not sufficient to extract user opinions in app reviews with intentions and topics specified by app developers (e.g., "extract all *suggestions* for *new songs* in 'Magic Title 3' game"). In this paper, we propose ALPACA , a new approach for this problem. ALPACA allows developers to specify their topics of interest via *keywords* and ALPACA uses *linguistic patterns* to match and extract phrases in app reviews containing the corresponding keywords.

Conference'17, July 2017, Washington, DC, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

A linguistic pattern is a commonly used grammatical structure. For example, "[somebody] should [do] [something]" is a linguistic pattern used for suggestions. The state-of-the-art tool SURF has used linguistic patterns to effectively extract user opinions and classify their intentions in app reviews. For example, the app review *"You should add some Hindi songs"* matches the aforementioned pattern and would be an excellent suggestion for the development team of 'Magic Title 3'.

The authors of SURF use a pre-defined set of linguistic patterns and hard-code them in SURF in the extraction procedure. This leads to several limitations. First, because the patterns are generic, each is hard-coded as a separate matching function. The input text is parsed into a parse tree each pattern matching function is called against that parse tree to determine if the text matches the corresponding pattern. Thus, SURF's pattern matching is slow. More importantly, users often use slightly different patterns to describe the same idea. However, the pattern set of SURF is fixed and hard-coded, thus, could only extract text matches exactly with the hard-coded patterns, leading to a low recall rate in SURF's classification (e.g., recall rate for extracting feature requests is 22.5% [24]).

ALPACA has several advantages over SURF. First, rather than using generic patterns like "[somebody] should [do] [something]", ALPACA uses more restricted ones , such that "PR should COMP-VB COMP-NN". In that pattern, PR is the placeholder for a pronoun, COMP-VB is for a simple verb phase, and COMP-NN is for a simple noun phase. PR, COMP-VB, and COMP-NN are defined based on part-of-speech (PoS) tags. ALPACA defines its linguistic patterns as sequences of tokens, each could be an actual functional word like 'should' or a placeholder. We have defined a similarity measurement for such linguistic patterns, thus, ALPACA can expand its set of patterns from input data. In addition, we design a near-matching algorithm to match ALPACA 's patterns to text. This algorithm is a sequence-based, thus, does not require parsed trees. Thus, it is much more efficient and effective.

Our experiments show that ALPACA is faster and has a higher recall rate than SURF. For example, it processes more than 10 thousands reviews of 'Magic Title 3' in around 7 minutes, while SURF needs nearly 3 hours. It has a recall rate of 87% while SURF can only archive 50.5%. In our case studies, app developers of three subject apps find that the opinions reported by ALPACA is more specific and useful that those reported by SURF.

Our technical contribution includes the following:

- A complete framework to extract opinions with specific topics and intentions using linguistic patterns and keywords.
- A flexible and expandable set of linguistic patterns
- A novel sequence-based, near-matching algorithm for matching linguistic patterns to text.
- A semi-automated method for expanding linguistic patterns

The remaining of this paper is organized as the following. Section 2 defines the concept of linguistic patterns and its similarity. Section 3 discusses details of our approach to extract user opinions. Section 4 evaluates our approach by comparing it to several base-lines and shows three case-studies. Section 5 discusses about the works that inspired our researches and the most related works in the field. Finally, the last section concludes this paper.

2 LINGUISTIC PATTERN DEFINITION AND SIMILARITY

In previous studies [24], linguistic patterns were used as a mean to discover certain intentions of reviews, i.e. *problem discovery, feature request, information giving, information seeking, solution proposal, opinion asking.* However, these linguistic patterns are not only vague to Natural Language Processing (NLP) techniques, but matching them require a hard coded matching algorithm without any flexibility. Moreover, it takes great human resource to discover new patterns manually. Therefore, in this section, we revise the definition of *linguistic pattern* in a more NLP friendly way, as well as introducing a novel measurement of similarity between patterns.

2.1 Definition of Linguistic Pattern

DEFINITION 1. **Functional Words** *are words that do not offer significant meaning to a text sequence in a specific domain. The list of Functional Words include but not limited to: 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 stop words (e.g. need, want, have, able, unable)*

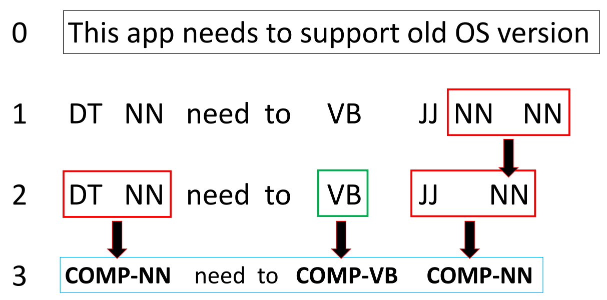
DEFINITION 2. **Part-Of-Speech Compositions** *are compositions of consecutive Part-Of-Speech (POS) tags following the rules given in our Appendix* [1]. *There are three possible compositions: COMP-JJ (adjective composition), COMP-NN (noun composition), COMP-VB (verb composition).*

DEFINITION 3. **Linguistic Pattern** *is a sequence of POS tags, compositions, and functional words that is grammatically sufficient and often indicates same intentions to all text sequences it represents.*

In English, we often use a generic pattern such as "[something] need [something]"[24] to describe a pattern of one particular intention (i.e. feature request). By replacing placeholders such as [something], with different words, we can create sentences of the same intention but convey different meaning and context. For current NLP tools [4, 9], the generic templates discussed above can potentially be matched using a tree traversal algorithm based on the Parse Tree of the sentence. However, this technique suffers greatly in performance due to the high complexity nature of tree traversing. Not just that, it also can be prone to low recall on sentences with typos and messy punctuation, which usually is the case for mobile app reviews [27]. More to that, the only current known implementation of such generic pattern to be readable to machine in DECA is to hard-code every parts of the parse tree of the pattern, which is inherently not scalable. Therefore, we have decided to approach these patterns on another perspective: *POS sequences.*

It is common practice to tag Part-of-Speech (POS) on every single word for later analysis in NLP. However, since each POS sequence are potentially varies even if they have the same linguistic patterns, it is not possible to detect all of them without generalizing them to a certain level. For example, in the pattern "[something] need [something]", the non-funtional part [something] could be one noun (NN), or a group of noun (NN NN NN), or a noun with one support adjective (JJ NN) or several support adjectives (JJ JJ NN), or the Noun could be in plural form (NNS), or even a verb phrase (to VB NN), and more to that.

Figure 1: **Example of pattern extraction from a clause**


0
This app needs to support old OS version
1
DT NN need to VB JJ NN NN
2
DT NN need to VB JJ NN
3
COMP-NN need to COMP-VB COMP-NN

To reduce this complication of POS tags sequence and generalize patterns, in ALPACA , we apply three level of abstractions on each POS tag sequence to create its final form: First, ALPACA stems the words into its root word form, ruling out the variant for verb, noun and adjectives (i.e. plural noun become noun, verbs with different tenses become just verb, and adverb with post-fix "ly" become just adjective.). Secondly, it groups the consecutively same POS tags of non-functional words into one single POS tag (e.g. NN NN NN become just NN, or ADJ ADJ become just ADJ). Lastly, our framework combines the different consecutive POS tags into their composition forms (i.e. COMP-VB, COMP-NN, COMP-JJ as shown in Definition 2.1) using several known linguistic rules extracted from English teaching sources[1, 2]. The rules are included in our Appendix [1]. Figure 1 shows an example of how a sentence be reduced to a linguistic pattern that matches to "[something] need [something]".
In conclusion, our linguistic pattern is defined as a POS tag sequence after the original POS tag sequence goes through several abstraction layers, while keeping the functional words.

2.2 Linguistic Pattern Similarity

As discussed above, a generic pattern can be something like "[something] need [something]" that has a specific intention. However, even this generic pattern can varies into slightly different patterns with the same intention as well (e.g. "[something] need [something] and/or [something]"). These kinds of variation could be limitless, despite of implying the same intention, which is a request. Further observation suggested that similar patterns often share a similar set of functional words (Definition 2.1) in the same order (e.g. "have to" and "to have" describe entirely different intentions) . Non-functional words usually can be replaced in the placeholders to create a new sentence, thus have little impact on how an intention is interpreted.

With this observation, we concluded that functional words and their order have big impact on the intention of the linguistic pattern. Therefore, by finding patterns of the same functional words order, we can find more patterns of similar intentions. Based on this, We designed a Linguistic Pattern Similarity measurement using JACCARD similarity [22] of sub-sequences of functional words from both patterns. The maximum length of a sub-sequence can be as long as the list of the functional words in both patterns, or reduced to two or three words for a more reasonable computational cost. Formula 1 shows how our measurement is calculated.

Note that this measurement is based on the assumption that most patterns with similar functional words order have the same intention. There are cases that the more general functional words can combine with non-functional words to create patterns that are too generic to be classified into one intention. Currently, to the best of our knowledge, there is no definite formula for the way English language is used. We call such incorrectly classified cases as errors. In the next subsection, we describe the process of eliminating such errors with human intervention. In section 4, we evaluate the performance of our measurement method with both human intervention and no-human intervention.

ϕ_i : Linguistic patterns i
ξ_i : set of functional subsequences belong to ϕ_i, in *Strict* version, this set include both functional and non-functional words

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

We can also extend this method by adding the list of placeholders to lessen the number of errors. However, doing so will reduce the potential pattern variation space as well. We called this a "Strict" version of our similarity measurement and applied it in the Near-Matching technique for its characteristics. More detail of this technique is discussed in Section 3.4.

3 OPINION EXTRACTION

DEFINITION 4. **User Opinion** *of a topic is a sequence of words within a review that matches to a linguistic pattern of an intention and contains at least one keyword of that topic.*

3.1 Overview of ALPACA

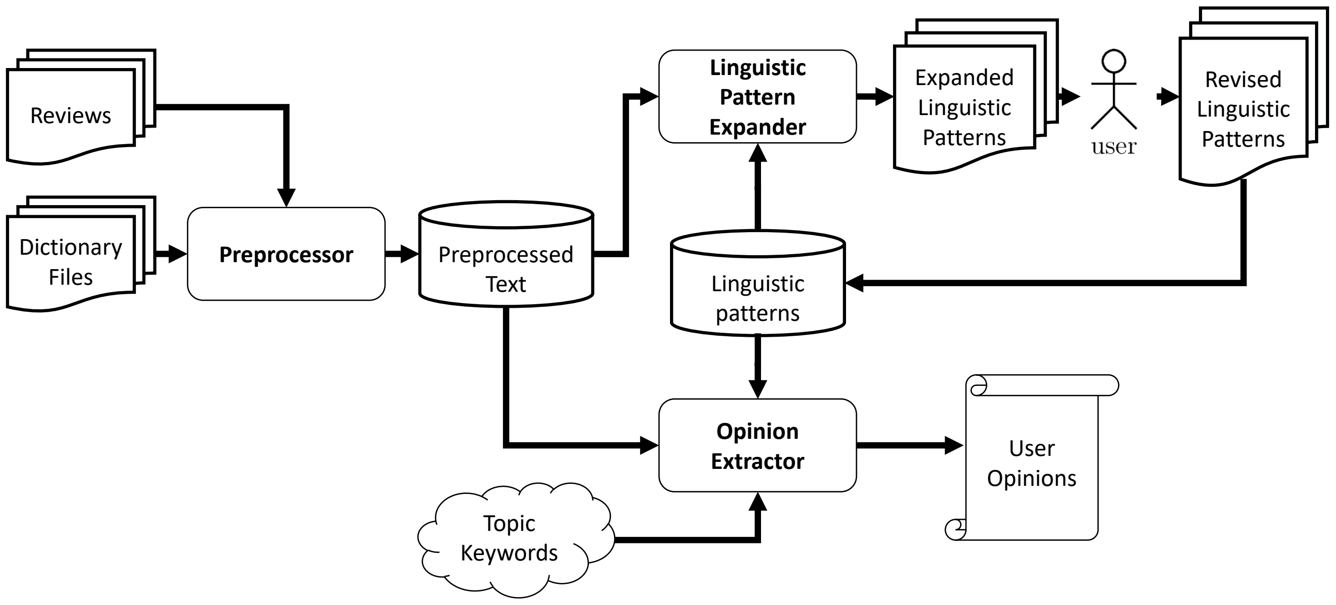
Opinion extraction for reviews has been discussed by Vu *et al.* and Panichella *et al.* in their previous works. In their work, respectively, they define an opinion as a group of keywords or a sentence belong to a topic and a intention. Both of them have their strong sides: MARK is able to find customized topic that developers want to know about, while SURF can classify a sentence into one of the 12 major intentions. However, SURF is limited to a few predefined topics and intentions, while MARK can not classify intention at all. In our case studies 4.4, both MARK and SURF could not answer the specific questions that were asked by app developers. In this section, we propose ALPACA as a robust approach to solve both problems, and give developers what they would want to know about the opinions from their users. Furthermore, we hereby redefine User Opinion as in Definition 3

As shown in Figure 2, ALPACA have two different use cases: Pattern Expansion and Opinion Extraction. We will discuss more about Pattern Expansion in Section 3.3. The input for each task module can be modified outside of the framework and will directly affect the results. Modifiable inputs includes: Dictionary files, Linguistic patterns, topic domain/functional words. These inputs are included as artifacts [1] for this paper, with the domain/functional words we used for reviews.

The output for Pattern Expander is a list of expanded patterns, 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 10

[1]Artifact link: https://goo.gl/VPuE9t

Figure 2: **Overview of ALPACA**



3.2 Preprocessing

App reviews are often messy [26]. Therefore, it is necessary to preprocess review text before moving further to other text mining phases. For the task of Opinion Mining, we choose to tune ALPACA to correct misspelled words and reduce them to root-words to avoid missing important keyword information from over-stemming. This process takes in inputs such as a predefined dictionary of root-words, a word corrector map, a list of functional words, and possibly app-specific words. We have provided this dictionary along with ALPACA as an artifact. In detail, the English root-words were extracted from WordNet3.0 [20]; the domain specific words for app reviews came from MARK [26] and Linux dictionary; the functional words (connectors, intensifiers, negations, wh) came from a previous study [27].

3.2.1 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[19]. 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.

3.2.2 Pattern extraction. Originally suggested by Vu *et al.* 's idea of extracting phrase template[27] for faster mapping of phrases, we expanded the concept of phrases into our concept of linguistic patterns. Within Preprocessor module, if enabled, the patterns are originally extracted using Stanford phrasal extraction [9], before put through three-level of abstraction process as defined in Section 2.1, all held together by functional words. Users can also add more functional words for their domain of interest, enabling the capturing of heuristic patterns for specific domains, which in this case could come from specific apps.

Despite of the slow performance for pattern 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 in the case of user reviews for mobile apps.

These patterns can be used as input for the optional Pattern Expansion module as a way to find patterns of a certain intention. We will discuss more about this module in the next Section.

Once user of ALPACA has the desired patterns, they can start the Near-Matching module, which takes input of user reviews, linguistic patterns and topic keywords to find text clauses that fit such patterns and contain at least one topic keyword. More on how pattern are matched using our Near-Matching technique is discussed in Section 3.4.

3.3 Pattern Expansion

In previous studies from Panichella *et al.* [24], linguistic patterns for the intentions of each sentence had to be manually extracted from a truth set of sentences. This process takes time and effort, and not scalable. Moreover, app users may have various way of expressing their own opinions in written form. Therefore, a small set of linguistic patterns may not be able to classify millions of different sentences from app reviews. Panichella *et al.* [10] attempted to overcome this problem by applying machine learning techniques on linguistic patterns founded by DECA. However, the underlying problem still remains, which is the limitation of a small set of patterns over a much larger potential pattern pool generated by users. Therefore, ALPACA incorporate a semi-automatted pattern expansion process, and a Near-Matching technique. In this subsection, we further explains how our approach work.

As discussed in Section 3, our technique requires an adequate amount of patterns for each intention to ensure good coverage of

```
function expandPattern
  Input: $O$ is a set of original linguistic patterns
         $P$ is a set of Phrases extracted with Stanford NLP
         $\omega$ threshold of pattern similarity
  Output: $R$ is a set of linguistic pattern that matched a pattern in $O$
  for each $p$ in $P$
    $\gamma$ = patternize($p$)
    for each $o$ in $O$
      if( patternSim($\gamma$,$o$) $\geq \omega$ )
        $R$.add($\gamma$)
  return $R$;
```

Figure 3: **Expanding patterns using a set of linguistic patterns on a set of phrases extracted by Stanford NLP**

the actual opinions in reviews. To get these patterns, we employed a method to expand similar patterns from a small set of pattern.

ALPACA has provided a way to extract linguistic patterns for reviews, however, these patterns are not following just a particular intention, but all the intentions there be. Meanwhile, DECA also provided a set of 246 patterns of several specific intentions which were manually obtained. With these original patterns, or any user provided set of patterns, ALPACA simply surfs through all the extracted patterns γ from reviews to find the ones that has similarity score above a threshold with at least one pattern in the set as shown in Algorithm 3. The threshold ω can be set by user.

However, the process of expansion even after our similarity measurement still does not guaranty the expanded patterns to strictly follow the intention of interest due to the various ways of expression in English. Without considering the meaning of the actual words in a clause, there is no sure way to tell the true intention. Some generic patterns can be used to express different intetions (e.g. "add a new pet or I rate 1 star" and "Orange or apple". Both have the same linguistic pattern (i.e. [something] or [something]), but the former describe a feature request, while the later is just giving some information. Those kinds of pattern need to be decided by human to be included in the final result or not to increase precision of the partern matching stage. For user's convenient, ALPACA also provides the examples of how original text looks like for each sentence. Moreover, it has already helped reducing the number of patterns to look at by our three-level abstraction discussed in section 2.1, but user can further reduce that number by adjusting the threshold.

To evaluate the performance of pattern expasion with and without human intervention, we designed an experiment that is further discussed in section 4.

3.4 Near-Matching

One limitation of DECA was their slow speed in nature when it comes to matching pattern since they use Stanford Parse Tree to match the structure of each sentence. Moreover, if the text has slight punctuation or typing mistakes, the parse tree will be changed drastically[27], resulting in a true negative classification.

By matching patterns that have already been mined from reviews on their POS tag form, we can eliminate the speed problem. However, if we only match the sequence with our pattern exactly (here by we call it Exact-Matching), it still is susceptible to typing mistakes and punctuation problem and effectively reduce recall.

To increase this recall, we apply the "strict version" of Pattern Similarity Measure as our matching condition instead of just match the sequence with exact patterns (here by we call it Near-Matching).

```
function nearMatch
  Input: $T$ is a list of linguistic patterns
         $S$ is an array of word, representing a sentence
         $\omega$ threshold of pattern similarity
  Output: $seeds$ is a list of subsequences that matched a pattern
  for (s = 0; s < $S$.length; s++)
    for (int e = $S$.length − 1; e >= s; e−−)
      if(isSubsequence(s,e,$seeds$))
        if (patternSim_Strict(s,e) >= $\omega$)
          $seeds$.add(s,e)
          break;
  return $seeds$;

function isSubsequence
  Input: $s$ is the start index of the subsequence
         $e$ is the end index of the subsequence
         $seeds$ is a list of subsequences
  Output: TRUE if no subsequence contains both s and e. FALSE otherwise
  for each $\alpha \in seeds$
    if($\alpha$.s <= s && $\alpha$.e >= e)
      return FALSE
  return TRUE
```

Figure 4: **Matching pattern with the Near-Matching algorithm**

As discussed in previous sections, regular Patterns Similarity Measure may require human intervention to decide which pattern is actually correct, thus effectively decrease the precision. "Strict version" would be the best of both worlds, and works without human.

Lastly, in each sentence, there could be multiple clauses that matches a pattern, and it is possible that the bigger clause contains a smaller clause that also matches a pattern. We simply choose the largest clause as it should also contains the information conveyed in the smaller ones. This process is described in Algorithm 4.

Table 5 and 2 compare the F-score using both Exact-Matching and Near-Matching. The detail is discussed in our evaluation.

4 EVALUATION

In this section, we evaluate and demonstrate sensitivity, effectiveness, and usefulness of ALPACA comparing to comparable baselines. Our evaluations include a comparison of classification power with DECA using both Near-Matching and exact matching; a test of F-score for pattern expansion with and without human intervention; and three use cases of how useful ALPACA is comparing to SURF with actual developers.

4.1 Sensitivity and effectiveness

To evaluate sensitivity and effectiveness of our pattern related techniques, we designed our tests to answer two research questions:

RQ1: How does ALPACA 's Linguistic Pattern Expansion perform comparing to DECA's original pattern set matching on the truth set?

RQ2: How does linguistic pattern Near-Matching technique perform compare to the exact-matching technique?

4.1.1 Dataset. For this evaluation, we use the truth set provided by Panichella *et al.* [10]. This truth set contains reviews classified into four intention classes: Problem Discovery, Feature Request, Information Seeking and Information Giving (Table 1).

In the next experiments, we use the 246 linguistic patterns from DECA as input for the expansion of intentions of interest, as well as other related experiments.

4.1.2 Evaluation design. To answer the two research questions above, we used DECA as the base line for classification to compare with our approach. Moreover, since Pattern Expansion broadens

Table 1: Overview of truthset

	No. of reviews
Information Seeking	101
Information Giving	603
Feature Request	192
Problem Discovery	494
Total	**1390**

the number of pattern, it should increase recall and decrease precision by nature. Therefore, we use F-score as our main comparison measurement. With this in mind, we kept expanding the patterns until F-score did not increase anymore. The original pattern set was taken from DECA pattern set to run directly on our matching algorithms for fair comparison.

In addition to increasing the number of iteration for expanding pattern, we also ran the two matching methods on every iteration for a fair comparison. Note that the original set does not need human intervention, therefore, we did not provide results for it.

For human intervention to filter out patterns that were not fit in the intention of interest after each expansion, the author read through each pattern and its example. If the example does not clearly represent the pattern of interest, we eliminate that pattern.

Within the scope of this evaluation, we only focus on Problem Discovery and Feature Request as our case studies (Section 4.4) suggested that developers have a reasonable interest in those two intentions. Moreover, since the nature of intentions does not dictate our definition of Linguistic Patterns, there is no reason to test more.

Table 2 shows the results of our experiments as described.

4.1.3 RQ1: How does ALPACA 's Linguistic Pattern Expansion perform comparing to DECA's original pattern set matching on the truth set? For the case of Problem Discovery intention, as shown in Table 2, we can see their F-score has already high with Stanford Parse Tree matching (about 75%). However, the same pattern set did not fare well to our matching algorithm. This is reasonable, as our definition of Linguistic Pattern is different from theirs in nature. However, we can see that after the third iteration of expanding those same pattern, especially with human intervention, we see a significant increase in both recall and precision. From the third iteration onward, our approach started to have higher F-score than DECA's, capping at about 80% at iteration five. At iteration six, our recall reached 85% comparing to DECA at 72%.

For the case of Feature Request intention (Table 5), DECA had only 36% total in F-score. Even without any iteration, the Near-Matching approach had already reached 40%. The highest score was given to Near-Matching with human intervention after the first iteration at 56%. This might be due to the fact that feature requests patterns are not outright distinctive to other patterns, hence the low precision. On Table 6, we can see that our approach out performed DECA significantly, reaching 87% recall on iteration 3 without human intervention.

Figure 5: **F-score of Problem Discovery classification with different thresholds over 6 iterations**

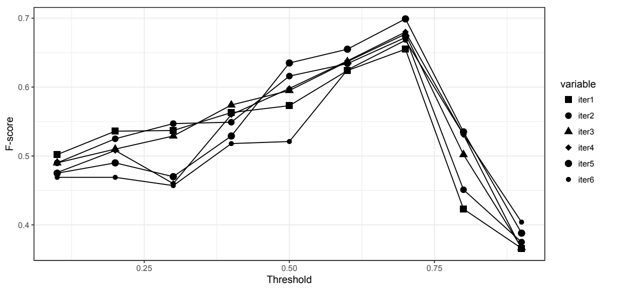
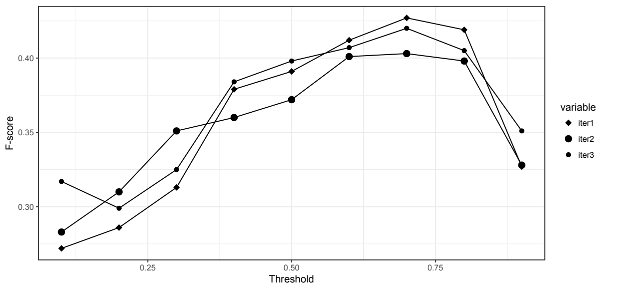


Figure 6: **F-score of Feature Request classification with different thresholds over 3 iterations**



Overall, our technique worked consistently better than DECA on the same truth set after a few expansion iterations without having to spend computational power for traversing the Parse Trees.

4.1.4 RQ2: How does linguistic pattern Near-Matching technique perform compare to the Exact-Matching technique? On both Table 2 and Table 5, Near-Matching approach often reached maximum F-score one iteration sooner than Exact-Matching, with or without human intervention. This is a quite surprising result, since Near-Matching is a trade off between higher recall and lower precision. We would expect it to have a similar F-score to Exact-Matching. This is an indication that the increase in recall out-weighted the decrease of precision, making Near-Matching seemingly superior to Exact-Matching on the truth set, especially with human intervention.

In conclusion, Near-Matching technique is consistently about as effective as or even more than Exact-Matching on our truth set. Moreover, since manually filtering out unrelated patterns often takes time and effort, this approach would be more valuable without compromising classification power.

4.2 Similarity Threshold Evaluation

In this section, we evaluate the effect of different thresholds on the result for expansion. It is noteworthy that the other experiments in Section 4.1 are using a similarity threshold of **0.7**. The reason was that this threshold gave the best results for the truth set, as shown in Figure 5.

For this experiment, we apply different thresholds with an increment of 0.1 each step for both Problem Discovery and Feature

Table 2: F-score comparison of ALPACA with DECA on Problem Discovery intention

		DECA	Original patterns	iter 1	iter 2	iter 3	iter 4	iter 5	iter 6
Without human	Exact Match	0.749	0.271	0.655	0.672	0.677	0.68	**0.699**	0.668
	Near Match	0.749	0.357	0.656	0.672	0.68	**0.692**	0.687	0.655
With human	Exact Match			0.684	0.733	0.761	0.781	**0.798**	0.684
	Near Match			0.696	0.739	0.756	**0.799**	0.777	0.74

Table 3: Recall comparison of ALPACA with DECA on Problem Discovery intention

		DECA	Original patterns	iter 1	iter 2	iter 3	iter 4	iter 5	iter 6
Without human	Exact Match	0.72	0.158	0.728	0.759	0.767	0.773	0.817	**0.846**
	Near Match	0.72	0.223	0.735	0.759	0.773	0.795	0.832	**0.852**
With human	Exact Match			0.568	0.657	0.702	0.734	0.767	**0.781**
	Near Match			0.613	0.669	0.706	0.748	0.769	**0.805**

Table 4: Precision comparison of ALPACA with DECA on Problem Discovery intention

		DECA	Original patterns	iter 1	iter 2	iter 3	iter 4	iter 5	iter 6
Without human	Exact Match	0.78	**0.951**	0.595	0.603	0.606	0.607	**0.61**	0.552
	Near Match	0.78	**0.887**	0.593	0.603	0.608	**0.613**	0.586	0.532
With human	Exact Match			0.859	0.831	0.832	**0.836**	0.833	0.74
	Near Match			0.805	0.825	0.813	**0.856**	0.785	0.76

Table 5: F-score comparison of ALPACA with DECA on Feature Request intention

		DECA	Original patterns	iter 1	iter 2	iter 3
Without human	Exact Match	0.359	0.157	**0.427**	0.403	0.42
	Near Match	0.359	0.403	**0.423**	0.403	0.379
With human	Exact Match			0.514	**0.55**	0.459
	Near Match			**0.559**	0.555	0.435

Request intentions. For each increment, we run Problem Discovery intention through six iterations and Feature Request through three iteration using Exact-match. The reason being our results showed that after those number of iteration, F-score did not increase anymore, and Near-Match would operate on similar principles to expanding. Moreover, please note that we only apply the expansion process without human intervention for this experiment, since filtering with human labor can be subjective, and it is costly. Lastly, since it is obvious that lower threshold would result in higher recall and lower precision, we only account F-score for this test.

The results are shown in Figure 5 and Figure 6. These results suggest that the threshold *0.7* seems to yield highest F-score consistently for both kinds of intention. The reason for a sudden decrease in F-score of thresholds after 0.7 could be because of the low recall while the precision does not increase.

4.3 Running time evaluation

One of the major improvement with our approach is running time. As discussed, ALPACA chose a pattern matching approach because it is relatively faster than traveling through a Parse Tree. The question is: how faster is it?

In this evaluation, we compare ALPACA and DECA classification speed on the same computer on several different dataset. We chose not to include SURF is because it uses DECA underneath as one of its modules. Our computer is an i7 HP laptop with 12GB of RAM. The datasets used are the truthset used in evaluation for sensitivity and effectiveness (Table 1) and the reviews data from our case studies (Table 9). Please note that we have included the preprocessing time for ALPACA in this experiment.

Table 6: Recall comparison of ALPACA with DECA on Feature Request intention

		DECA	**Original patterns**	**iter 1**	**iter 2**	**iter 3**
Without human	Exact Match	0.505	0.089	0.677	0.792	**0.828**
	Near Match	0.505	0.522	0.724	0.823	**0.87**
With human	Exact Match			0.521	0.656	**0.781**
	Near Match			0.599	0.682	**0.797**

Table 7: Precision comparison of ALPACA with DECA on Feature Request intention

		DECA	**Original patterns**	**iter 1**	**iter 2**	**iter 3**
Without human	Exact Match	0.278	**0.708**	**0.312**	0.301	0.281
	Near Match	0.278	**0.386**	**0.299**	0.297	0.242
With human	Exact Match			**0.508**	0.474	0.325
	Near Match			**0.523**	0.468	0.299

The result of this evaluation is shown in Table 8. It is very straight forward that our approach is considerably faster than the Parse Tree approach, thus validating our initial assumption.

4.4 Case studies for usefulness

This sub-section evaluates the usefulness of ALPACA with three case-studies: Magic Tiles 3, Tankraid, and Face Dance Challenge.

4.4.1 Case studies setup. In each case study, we first interviewed the developers of the respective app to find out what they need, then we evaluate the usefulness of ALPACA for their app.

To evaluate the usefulness of ALPACA , we compared our results with SURF, since it has the most similar use-cases to ours. Both tools generate reports from the same datasets provided by developers of each app in the use-case. The dataset statistic can be found in Table 9. In order to keep this comparison fair, we copied the content reports from both tools into two different Microsoft Excel files in similar format and named them anonymously. Each file separates the topics into different tabs. Within each tab, we listed the founded reviews and its classified intention. For ALPACA , we also highlighted the part of text that matched our linguistic patterns as it is part of our functionality. These reports were then sent to each developer to ask them which one is more useful to them and more of their feedback. The results will be discussed in detail in each case study section bellow.

For each app, we also discuss the processing speed of ALPACA comparing to SURF on our i7 experimental laptop.

4.4.2 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. We reached out to their developers in November 2017 to interview them and they were nice enough to give us the review dataset to test our ALPACA on. During our direct interview, we have learned that the developers want 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 MARK[26] to find the keywords for those topics and ran them on their 10,453 English reviews, the whole process took 4 minutes after having the keywords (The example report can be found in Table 10). With the same amount of text on the same computer, SURF took 183 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 ALPACA gave them the information they asked for. In other word, both are useful to the developers.

4.4.3 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.

We have interviewed the developers of Face Dance Challenge in November 2017 to learn about what aspect they care the most about the game. At first, the developers simply wanted to know what users want to upgrade in the game (i.e. feature requests), therefore, we ran ALPACA without topic on their reviews. The results were various in topic, and we have 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.

With the topics, ALPACA took 2 minutes to run through 5,095 English reviews to produce the report. SURF took 96 minutes for the same task. The actual report can be found in our Appendix.

Table 8: Time required for classification using ALPACA and DECA on different datasets (in minutes)

	SURF truthset 1390 reviews	**Tankraid** 1648 reviews	**Face Dance** 5094 reviews	**Magic Tiles 3** 10453 reviews
ALPACA	1m 1s	1m 7s	3m 38s	6m 51s
DECA	23m	25m	1h 34m	2h 58m

Table 9: Reviews dataset for three case studies

	No. reviews	**Date Ranges**
Face Dance Challenge	5094	Aug - Nov, 2017
Tankraid	1648	Jan - Nov, 2017
Magic Tiles 3	10453	Mar - Nov, 2017

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 have concluded that this topic is simply "feature request", and ALPACA can simply list all of which without any topic keyword.

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

4.4.4 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 ALPACA specifically for problem discovery without any topic and reported it back to them. They have verified to us that the founded bugs were consistent with what they have found by themselves. Therefore, we have extracted the topic from this finding and classified them into three main topics: Control, Battle and Match Making, In-app-purchase.

ALPACA took less than 1 minute to ran through 1,648 reviews to generate the report, while SURF took 27 minutes. The example report by ALPACA can be found in Appendix.

After carefully comparing two reports, developers of Tank Raid thought both are useful to them. However, they liked the report with highlighted opinions 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, ALPACA is necessary to them.

5 RELATED WORK

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

In [25], 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.[14] 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.[23]. Similarly, Khalid et al. suggest that there are at least 12 types of complaints about iOS apps[17]. 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.

Other than reviews, price and rating of apps can also affect how people provide their feedbacks, as suggested in [16] by Iacob et al. Meanwhile, Bavota et al.[3] studied the relationship between API changes and their faulty level with app ratings. Recently, Martin et al. [18] 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[13]. extracts features from app reviews in form of collocations and summarizes them with Latent Dirichlet Allocation (LDA)[5] and their sentiment.

Some other works resulted in complete tool set or prototypes such as Wiscom [11] or MARA[15], or AR-Miner [8]. 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)[21] to classify between informative and non-informative reviews. To rank the reviews, they use a ranking schema based on several meta-data of reviews and try to suggest the most informative ones.

Later on, MARK [28] proposed a keyword based approach to discovering topics and trends using their semantic meaning. However, MARK did not try to capture the intention of user on their reviews.

The most closely related work to ALPACA 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[24], 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 linguistic 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, ALPACA framework provides an unbounded, flexible, and robust

Table 10: Example report for Magic Tiles 3

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

Table 11: Example report of Face Dance Challenge

Review	Opinion	Intention	Topic
Can it **be played by two people in one camera**? Cause **I would like to try this with my girlfriend**	. be play by two people in one camera . I would like to try this with my girlfriend	Feature 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..	. but please make the control of the face easy	Feature 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.	. still get perfect score without move your face	problem discovery	Face Detection and Camera
Its so **cool I wish there was a booty dance challenge**	. cool i wish there be a booty dance challenge	Feature Request	Social
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.**	. try share it to facebook or something or send the video to my friend and the file will say it be corrupt	problem discovery	Social
I dont know why it's showing free game but this is paid game. **They will force you to buy song** and after that you can play this game. There is a upload song option but seriously it doesn't work.	they will force you to buy song	problem discovery	In App Purchase

environment for opinions discovering on user reviews using the novelty approach of redefined linguistic patterns.

6 CONCLUSION

In this paper, we have discussed the need of app developers for a new tool that can extract user opinions from app reviews by both topic and intention. We have also discussed how the current state-of-the-art tools are not capable of meeting this need.

To address this, we have introduced ALPACA as a new tool to extract user opinions based on their topics and intention. Our approach requires the definition of a new type of linguistic pattern, and led to a novel method for pattern similarity. With this approach, ALPACA can expand linguistic patterns and greatly reduce the manual effort for discovering them. Moreover, it also enables us to develop the Near-Matching algorithm that does not rely on a parse tree of the text, nor be hindered by slight differences in the way users write their words. This speed up the classification speed greatly comparing to DECA, a tool that uses parse trees to classify intentions.

Finally, we have evaluated our approach on multiple sides: comparing to SURF in term of sensitivity, effectiveness; comparing classification time to DECA; evaluate different thresholds of similarity, and conducted three case studies with real app developers. Our evaluation has shown that ALPACA is not only faster, more sensitive than previous tools but also is useful to developers.

REFERENCES

[1] Compound Nouns. (????). https://www.learnenglish.de/grammar/nouncompound.html
[2] Compound Verbs. (????). http://grammar.yourdictionary.com/parts-of-speech/verbs/compound-verb.html

[3] G. Bavota, M. Linares-Vasquez, C.E. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. 2015. The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps. *Software Engineering, IEEE Transactions on* 41, 4 (April 2015), 384–407. DOI:http://dx.doi.org/10.1109/TSE.2014.2367027

[4] Steven Bird. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*.

[5] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (March 2003), 993–1022. http://dl.acm.org/citation.cfm?id=944919.944937

[6] Rishi Chandy and Haijie Gu. 2012. Identifying Spam in the iOS App Store. In *Proceedings of the 2Nd Joint WICOW/AIRWeb Workshop on Web Quality (WebQuality '12)*. ACM, New York, NY, USA, 56–59. DOI:http://dx.doi.org/10.1145/2184305.2184317

[7] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In *ICSE'14*.

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

[9] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, and others. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, Vol. 6. 449–454.

[10] Andrea Di Sorbo, Sebastiano Panichella, Carol V Alexandru, Junji Shimagaki, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *SIGSOFT'16*.

[11] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. 2013. Why People Hate Your App: Making Sense of User Feedback in a Mobile App Store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 1276–1284. DOI:http://dx.doi.org/10.1145/2487575.2488202

[12] L.V. Galvis Carreno and K. Winbladh. 2013. Analysis of user comments: An approach for software requirements evolution. In *Software Engineering (ICSE), 2013 35th International Conference on*. 582–591. DOI:http://dx.doi.org/10.1109/ICSE.2013.6606604

[13] Emitza Guzman and Walid Maalej. 2014. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*. IEEE, 153–162.

[14] Leonard Hoon, Rajesh Vasa, Jean-Guy Schneider, and John Grundy. 2013. *An Analysis of the Mobile App Review Landscape: Trends and Implications.* Technical Report. Tech. rep., Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia.

[15] Claudia Iacob and Rachel Harrison. 2013. Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 41–44. http://dl.acm.org/citation.cfm?id=2487085.2487094

[16] Claudia Iacob, Varsha Veerappa, and Rachel Harrison. 2013. What are you complaining about?: a study of online reviews of mobile applications. In *Proceedings of the 27th International BCS Human Computer Interaction Conference*. British Computer Society, 29.

[17] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed Hassan. 2014. What do mobile app users complain about? A study on free iOS apps. (2014).

[18] William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang. The App Sampling Problem for App Store Mining. (????).

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR* (????).

[20] George Miller and Christiane Fellbaum. 1998. Wordnet: An electronic lexical database. (1998).

[21] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. 2000. Text classification from labeled and unlabeled documents using EM. *Machine learning* 39, 2-3 (2000), 103–134.

[22] Gary W Oehlert. 2000. *A first course in design and analysis of experiments.* Vol. 1. WH Freeman New York. http://users.stat.umn.edu/~gary/book/fcdae.pdf

[23] D. Pagano and W. Maalej. 2013. User feedback in the appstore: An empirical study. In *Requirements Engineering Conference (RE), 2013 21st IEEE International.* 125–134. DOI:http://dx.doi.org/10.1109/RE.2013.6636712

[24] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *ICSME'15*.

[25] Rajesh Vasa, Leonard Hoon, Kon Mouzakis, and Akihiro Noguchi. 2012. A Preliminary Analysis of Mobile App User Reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference (OzCHI '12)*. ACM, New York, NY, USA, 241–244. DOI:http://dx.doi.org/10.1145/2414536.2414577

[26] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. Mining User Opinions in Mobile App Reviews: A Keyword-Based Approach (T). In *ASE'15*.

[27] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. Phrase-based Extraction of User Opinions in Mobile App Reviews. In *ASE'16*.

[28] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. Tool Support for Analyzing Mobile App Reviews. In *ASE'15*.