An Approach of Extracting Feature Requests from App Reviews

Zhenlian Peng[1,2], Jian Wang[1(✉)], Keqing He[1], and Mingdong Tang[2]

[1] State Key Lab of Software Engineering, Computer School, Wuhan University, Wuhan, China
{zlpeng,jianwang,hekeqing}@whu.edu.cn

[2] Computer School, Hunan University of Science and Technology, Xiangtan, China
mdtang@hnust.edu.cn

Abstract. With the rapid development of mobile technologies, developing high-quality mobile apps becomes increasingly important. App reviews, which are collaboratively collected from various users, are viewed as important sources for enhancing or evolving mobile apps, wherein how to accurately extract feature requests becomes an important issue. However, the scale of app reviews is so large that it is intractable to manually identify feature requests from these reviews. In this paper, we propose a semi-automated approach to extract feature requests based on machine learning approaches. In our approach, we firstly identify reviews on feature requests by defining suitable classification features and selecting appropriate classification approaches. Afterwards, these identified reviews are clustered using topic models, and phrases are extracted as feature requests, which serve as the basis of feature modeling. Experiments conducted on a real world data set show that the proposed approach can contribute to extracting feature requests from app reviews.

Keywords: Feature requests · App review · Classification · Word dependencies

1 Introduction

With the rapid development of mobile technologies, an increasing number of mobile apps have been developed and published. Similar to the traditional software development, the development of mobile apps also starts from requirements elicitation, where the quality of requirements plays a key role to assure the success of the software [1]. Since the approach of feature-oriented domain analysis (FODA) is proposed [2], the feature-oriented approach has been widely used in software development by software practitioners. According to IEEE standard glossary of software engineering terminology [3], a feature is defined as "a software characteristic specified or implied by requirements documentation". Due to the close relationship between requirements and features, extracting appropriate features will contribute to requirements elicitation, which will in turn promote the success of software development.

Various sources can be used to extract feature requests. For example, online open forums have been used to elicit features by project managers [4,5]. Domain knowledge coming from domain experts can also be utilized to elicit features by recommending proper expert stakeholders [6,7]. In addition, descriptions of online software products can also be leveraged to elicit software features [8,9]. In particular, reviews can be viewed as a way for users to collaboratively propose feature requests for a certain mobile app. So many works have been conducted towards extracting feature requests from app reviews. For example, an unsupervised information extraction system named OPINES is proposed in [10], which builds a model of important product features by mining reviews. Various information retrieval techniques such as topic modeling are leveraged to extract topics and representative sentences of those topics from user comments, which will be used to revise requirements for next releases of software [11]. A prototype named mobile app review analyzer (MARA) for automatic retrieval of mobile app feature requests from app reviews is designed in [12]. As a whole, these approaches mainly leverage information retrieval techniques in identifying feature requests from user reviews and they do not classify app reviews in advance. An automated approach that helps developers filter, aggregate, and analyze user reviews is proposed in [13]. However, they mainly focus on sentiment analysis on reviews and feature requests mining is not the focus of their work. Several classification algorithms are compared in [14] to classify app reviews into four types: bug reports, feature requests, user experiences, and ratings. They comprehensively classify app reviews, but they do not consider the characteristics such as linguistic rules that are specific to feature requests.

An approach of extracting feature requests from app reviews is presented in this paper. We focus on how to select appropriate classification attributes and an optimal classification algorithm to identify feature requests from the app reviews. Specifically, various classification attributes such as bag of words, linguistic rules and metadata (e.g., rating, tenses and sentiment) are analyzed. Four classification algorithms including J48, Naive Bayes, Random Forest [15] and SVM (Support Vector Machine) [16] are compared to select an optimal classifier. Then LDA (latent Dirichlet allocation) [17] is used to cluster reviews on feature requests. Finally, phrases that represent feature requests are extracted by using the Stanford Parser [18], a tool that can generate word dependencies of sentences, based on the clustered topics and terms.

The main contributions of our work are as follows.

- An approach of extracting feature requests from app reviews is proposed. Various possible selected classification attributes from raw reviews and classification algorithms are discussed. In addition, we use LDA to cluster reviews on feature requests into various groups. Word dependencies are used to extract phrases that represent feature requests based on the clustered result.
- Experiments on a real world data set are conducted to identify reviews on feature requests and extract representative feature requests.

The remainder of the paper is organized as follows. Section 2 introduces feature requests extraction approach in detail. The evaluation of the proposed approach is discussed in Sect. 3. Section 4 discusses related work and we conclude the paper in Sect. 5.

2 Feature Requests Extraction

2.1 Overview of the Approach

An overview of the approach is described in Fig. 1, which mainly involves three steps:

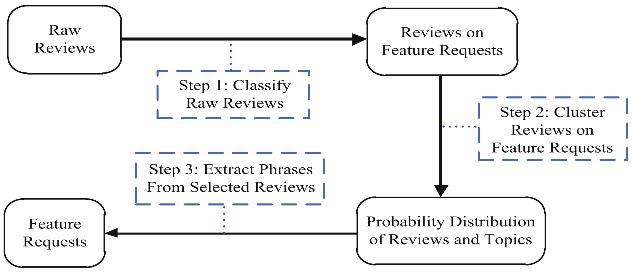
Raw Reviews
Reviews on Feature Requests
Step 1: Classify Raw Reviews
Step 2: Cluster Reviews on Feature Requests
Step 3: Extract Phrases From Selected Reviews
Feature Requests
Probability Distribution of Reviews and Topics

Fig. 1. Overview of feature requests extraction

Step 1: The objective of this step is to identify which reviews belong to feature requests using classification techniques. As depicted in Fig. 1, a classifier is trained by selecting appropriate classification attributes from these reviews. Then the classifier is utilized to predict whether unlabeled reviews belong to feature requests. To improve the prediction performance, it is important to select appropriate classification algorithms together with attributes.

Step 2: After raw reviews are classified, reviews on feature requests are clustered into semantically similar groups. Clustering algorithms have been widely used in mining features in the field of feature model extraction [8,9] and discovering Web service from text descriptions [19,20]. In this paper, LDA, a widely used topic model, is adopted to cluster these reviews based on identified latent topics.

Step 3: For each topic, the highly relevant reviews are selected, and then verb-noun phrases and noun phrases are extracted by analyzing from word dependencies of these selected reviews using the Stanford Parser. Finally, the phrases are filtered and selected as feature requests based on the relevance between its contained terms with the topic.

The extracted feature requests can be viewed as new requests of the mobile app, and therefore they can also serve as a basis of the mobile app evolution and the feature model change, which means that they will be developed or reused in the next release of the mobile app. Due to the space limitation, in this paper, we focus more on selecting appropriate classification algorithms and classification attributes.

2.2 Classification Attributes Selection

A review extracted from the Apple Store and the Google Play store [21] usually consists of the following attributes: app Id, review Id, review title, review comment, rating, reviewer, fee, date, and data source. However, not all the attributes are useful to train a classifier on identifying feature requests. Therefore it is necessary to select useful information from the review.

The title and comments are basic attributes of an app review, which can be treated as a document. In the document classification, bag of words (abbr. **BW**) are the basic classification attributes. The vectorization process of **BW** is usually described as follows: firstly a dictionary which includes all terms of reviews in the corpus is created; next, whether a term appears in the review and how often it appears is counted; and finally the *TF-IDF* of each term in a review is calculated. Some natural language processing techniques such as stop words removal and lemmatization are usually used during the process. In this paper, **BW** refers to bag of words together with stop words removal and lemmatization.

According to the analysis of the manually identified feature requests in the random sample described in [12], some keywords used for defining linguistic rules on feature requests have been identified in the title or comments. In order to reflect linguistic rules by using these keywords, they are classified into three categories: modal verbs (*abbr.* **MV**), general verbs or nouns (*abbr.***VN**), and preposition phrase (*abbr.* **PP**), as shown in Table 1.

Table 1. Keywords in reviews on feature requests

Part of speech	Keywords
MV	could, maybe, must, need, should, will, wish, want, would, please
VN	add, allow, complaint, hope, improve, lack, look forward to, miss, prefer, request, suggest, wait for
PP	if only, instead of

TF-IDF of each category is calculated to quantify the textual attributes. *TF* of a category in a review is the ratio between the number of keywords of the category occurred in the review and the total number of words in the review. *IDF* of a category in a review is a logarithm between the number of all reviews and the number of reviews containing any keyword of this category. *TF-IDF* of a category is the product of the *TF* and *IDF* score of the category. They are calculated using Eqs. (1), (2) and (3), respectively.

$$TF(c,r) = \frac{\sum_{k \in c} \#\, of\, k\, occurs\, in\, r}{\#\, of\, words\, in\, r}, \tag{1}$$

$$IDF(c) = log \frac{\#\, of\, reviews}{\sum_{k \in c} \#\, of\, reviews\, containing\, k}, \tag{2}$$

$$TF\text{-}IDF(c,r) = TF(c,r) \times IDF(c), \tag{3}$$

where, k represents a keyword in a category c (**MV**, **VN**, or **PP**) and r represents a review. In addition, we use **LR** to represent the combination of **MV**, **VN**, and **PP**.

The metadata such as star **rating**, **tenses** of the verbs, and reviewer **sentiment** can be extracted from app reviews. The star **rating** is a numeric value between 1 and 5 given by the reviewer, which will be used as a classification attribute. The **tenses** of verbs which occur in the review is also selected as a classification attribute because the future tense reflects a larger possibility on an enhancement of the app or a new feature request. Different from [14] which used past, present, and future tenses by part of speech tagging provided in NLP libraries, we only distinguish the future tense and the non-future tense in this paper. The reviewer **sentiment** reflects the positive and negative emotions of the reviewer [13]. Thelwall et al. [22] propose a fine-grained sentiment extraction approach, where one negative sentiment score in a scale of -5 to -1 and one positive score in a scale of 1 to 5 are assigned for each review. Similar to [13,14], an absolute score combined by negative and positive scores is used as a classification attribute.

2.3 Feature Requests Clustering and Extraction

As one of the most widely used topic models, LDA can be used to extract unobserved factors that capture the underlying domain semantics within the given documents. Once the reviews on feature requests are identified, LDA is leveraged to cluster these reviews on feature requests and identify the latent topics among them. More specifically, according to the distribution of topics in these reviews and the distribution of terms in topics generated by LDA, we can cluster the reviews where the highly relevant reviews on each topic are grouped together and we can also identify the highly relevant terms of each topic.

In our opinion, the feature requests can be represented in the form of verb-noun phrases, e.g.,"update screen", or noun phrases, e.g., "picture upload". Next, we pay our attention to extracting this kind of feature requests from the clustered reviews.

Inspired by our previous work on service goal extraction [23], in this paper we leverage the Stanford Parser [18] to extract the feature requests. The Stanford Parser can be used to perform linguistic analysis of sentences contained by reviews. The main linguistic analysis result we used is word dependencies, which describe the binary relations between words within a sentence. For example, amod(option-2, File-1) and compound(upload-6, picture-5) are two word dependencies in the review "File option such as picture upload will be loved". Based on these two word dependencies, we can get two potential feature requests: *file option* and *picture upload*. The Stanford Parser provides about 50 word dependencies, where we currently use a subset of them to extract feature requests, as shown in Table 2.

For each topic, we can extract feature requests from the reviews that are highly relevant to the topic using the above mentioned approach. Note that not all the phrases extracted from the word dependencies will be appropriate feature

Table 2. Used word dependencies

Dependency	Definition	Usage
dobj	A noun phrase which is the (accusative) object of the verb	Identify a "verb-noun" pair in an active clause
nsubjpass	A noun phrase which is the syntactic subject of a passive clause	Identify a "verb-noun" pair in a passive clause
nn(compound)	Any noun that serves to modify the head noun	Identify a "noun-noun" pair in a clause
amod	Any adjectival phrase that serves to modify the meaning of the noun phrase	Identify a "adj-noun" pair in a clause

requests relevant to the topic. A phrase can be viewed as a candidate feature request relevant to a topic if and only if it contains at least one highly relevant term of that topic. These candidate feature requests can be ranked according to their frequencies in the topic and the probabilities of its contained terms over the topic. In this way, we can get the feature requests from the reviews.

3 Evaluation

In this section, we evaluated the proposed extraction approach by a series of experiments. All the experiments are conducted on a PC with 3.19 GHz Intel Core i3 CPU and 4 GB RAM, running Windows 7 OS.

3.1 Experiment Data

In the experiments, we used the data set provided in [14], which was extracted from the Apple AppStore[1] and the Google Play[2]. In the data set, each review has the attributes of comment text, title, app name, category, store, submission date, and username. Furthermore, the metadata such as star rating, tenses of verbs and sentiment of the reviews were also extracted. Moreover, the types of reviews were manually analyzed and labeled, which were set as the grounding truth. In the data set, due to the great effort of manually labeling, 1924 reviews were labeled, where 295 reviews were feature requests, 600 reviews were non-feature requests, and 1029 reviews were labeled as other types such as bug reports, user experiences and ratings.

3.2 Evaluation Indicator

In order to evaluate the performance of various classification algorithms under different classification attributes, the standard metrics *precision*, *recall* and *F-measure* are used. In this paper, *Precision* is the fraction of reviews that are correctly classified to feature requests. *Recall* is the fraction of reviews on

[1] https://itunes.apple.com/us/genre/ios/id36.
[2] https://play.google.com/store?hl=en.

feature requests which are classified correctly. *F-measure* is a harmonic mean function of *precision* and *recall*. They are calculated by Eqs. (4), (5), and (6), respectively.

$$precision = TP/(TP + FP), \tag{4}$$

$$recall = TP/(TP + FN), \tag{5}$$

$$F\text{-}measure = \frac{2 \times precision \times recall}{precision + recall}, \tag{6}$$

where, *TP* is the number of reviews that are classified as feature requests and actually are feature requests. *FP* is the number of reviews that are classified as feature requests but actually are not feature requests. *FN* is the number of reviews that are classified into non-feature requests but actually belong to feature requests.

3.3 Results and Analysis

Firstly, a group of experiments are conducted in order to evaluate the performance of various classification algorithms under different classification attributes. The values of *precision*, *recall*, *F-measure* and execution time are compared by using various classification algorithms such as Naive Bayes, SVM, J48, and Random Forest under different classification attributes. In order to reduce the sensitivity of the data, the mean values of *precision* (*abbr. pre*), *recall* (*abbr. rec*), *F-measure* (*abbr. F1*) and the execution time (*abbr.* time)under equal scale of training and testing data were calculated for five times. Every time 75% of all the reviews labeled as feature and non-feature requests were randomly selected as training data and the remaining 25% reviews were selected as testing data. The result of this group of experiments is shown in Table 3 (please note that rat, ten and sen denotes rating, tenses and sentiment respectively).

As can be seen from Table 3, if BW is solely used as the classification attribute, *precision*, *recall* and *F-measure* of each classification algorithm is on the scale of 63.5% to 70.9%, 52.7% to 68.9% and 60.5% to 69.4%, respectively. Wherein, SVM can get the best *precision*, but its *recall* is the smallest. Naive Bayes can achieve the best classification results on the whole and its executing time is the shortest. Random Forest is suboptimum on the whole but its executing time is the longest. If LR is solely used as the classification attribute, we find that *precision*,*recall* and *F -measure* of each classifier is about 57.7%, 60.8%, and 59.2%, respectively. It is obvious that the *F-measure* of using BW is generally superior to that of using LR, but the executing time is the opposite. The reason is that BW consists of much more classification information meanwhile it has far more dimensions to compute for training a classifier. If BW and LR are used as classification attributes together, *precision*, *recall* and *F-measure* of each classifier are on the scale of 66.7% to 76.8%, 58.1% to 75.7%, and 65.6% to 75.2%, respectively. Four classifiers have the similar comparison of classification results with only BW being used as the classification attribute. Each algorithm can get better results by using both of them to show that BW and LR can get mutual supplement for training a classifier.

Table 3. Comparison of various algorithms under different attributes

Attributes	Naive Bayes				SVM			
	pre	*rec*	*F1*	time/s	*pre*	*rec*	*F1*	time/s
BW	0.699	0.689	0.694	0.05	0.709	0.527	0.605	0.2
BW + rat	0.697	0.716	0.707	0.05	0.707	0.554	0.621	0.22
BW + rat + ten	0.74	0.76	0.75	0.05	0.758	0.642	0.695	0.23
BW + rat + ten + sen	0.76	0.77	0.765	0.05	0.778	0.662	0.715	0.26
LR	0.577	0.608	0.592	0.001	0.577	0.608	0.592	0.05
LR + rat	0.609	0.598	0.603	0.001	0.587	0.608	0.597	0.08
LR + rat + ten	0.632	0.618	0.625	0.001	0.736	0.519	0.609	0.09
LR + rat + ten + sen	0.639	0.622	0.63	0.001	0.75	0.527	0.619	0.18
BW + LR	0.747	0.757	0.752	0.05	0.768	0.581	0.662	0.26
BW + LR + rat	0.757	0.757	0.757	0.06	0.808	0.568	0.667	0.27
BW + LR + rat + ten	0.816	0.816	0.816	0.06	0.845	0.703	0.767	0.32
BW + LR + rat + ten + sen	**0.824**	**0.824**	**0.824**	**0.09**	0.852	0.703	0.77	0.4
Attributes	J48				Random Forest			
	pre	*rec*	*F1*	time/s	*pre*	*rec*	*F1*	time/s
BW	0.635	0.635	0.635	0.94	0.697	0.622	0.657	1.97
BW + rat	0.655	0.642	0.648	1	0.676	0.649	0.662	2.04
BW + rat + ten	0.737	0.651	0.691	1.06	0.73	0.622	0.672	2.88
BW + rat + ten + sen	0.742	0.662	0.7	1.1	0.771	0.73	0.75	3.01
LR	0.577	0.608	0.592	0.002	0.577	0.608	0.592	0.01
LR + rat	0.592	0.614	0.603	0.002	0.608	0.619	0.613	0.01
LR + rat + ten	0.706	0.681	0.693	0.003	0.687	0.622	0.652	0.02
LR + rat + ten + sen	0.718	0.689	0.703	0.01	0.71	0.662	0.685	0.03
BW + LR	0.749	0.584	0.656	0.89	0.667	0.649	0.658	1.78
BW + LR + rat	0.759	0.615	0.679	1.28	0.727	0.649	0.686	2.38
BW + LR + rat + ten	0.761	0.628	0.688	1.39	0.768	0.716	0.741	3.12
BW + LR + rat + ten + sen	0.772	0.643	0.702	2.18	0.809	0.743	0.775	5.74

Furthermore, adding the metadata such as rating, tenses and sentiment can effectively improve the performance of each classification algorithm. According to the results, using tenses can get better performance than using the other two because future tense and non-future tense are considered together. When LR and the metadata are used as classification attributes, J48 can get better *F-measure* than the other three since J48 is more suitable for the sample with the smaller size. SVM can often get the best *precision*, but its *recall* is almost the lowest, so its *F-measure* is not so good. The performance of Random Forest is rather moderate and its execution time is the longest. The performance of Naive Bayes is superior to other classifiers if BW is used as one of classification attributes. When BW, LR and metadata are used as classification attributes, Naive Bayes can get the best results and both of its *precision* and *recall* can reach 82.4%. Additionally, the executing time of Naive Bayes is the shortest.

Table 4. Some topics and representative terms

Terms \ Topics	1	2	3	4	5
1	download$^{(0.031)}$	game$^{(0.053}$	give$^{(0.032)}$	love$^{(0.042)}$	update$^{(0.077)}$
2	note$^{(0.023)}$	fix$^{(0.050)}$	add$^{(0.032)}$	file$^{(0.032)}$	work$^{(0.043)}$
3	make$^{(0.020)}$	time$^{(0.030)}$	star$^{(0.027)}$	option$^{(0.027)}$	screen$^{(0.026)}$
4	version$^{(0.020)}$	play$^{(0.026)}$	feature$^{(0.021)}$	picture$^{(0.020)}$	open$^{(0.026)}$
5	way$^{(0.017)}$	phone$^{(0.026)}$	button$^{(0.019)}$	upload$^{(0.018)}$	video$^{(0.023)}$

Afterwards, LDA is conducted on the reviews on feature requests. Table 4 depicts some topics and top five representative terms in each topic. Wherein, each column represents a topic and each row shows terms and their probabilities in the corresponding topics.

Finally, for each topic, phrases are extracted from the clustered reviews by using the Stanford Parser. The top ranked feature requests in the data set are as follows: *update screen*, *fix game*, *game phone*, *add button*, *file download*, *open video*, *file option*, *picture upload*, *easy way*, and *proper version*. Clearly most of the identified feature requests are meaningful and can help requirements analysts in identifying new evolution requirements from app reviews. On the other hand, some resulting feature requests such as *easy way* and *proper version* are not satisfactory. How to further improve the identified feature requests using more word dependencies and more filtering rules will be our future work.

3.4 Threats to Validity

With respect to the internal validity, the main threat is that the proposed approach mainly considers various classification algorithms under different classification attributes for identifying whether a review belongs to feature requests. But it is also important to select proper clustering algorithms for grouping similar reviews into feature requests.

Threats to external validity concern the selection of keywords that occur in the reviews on feature requests. Since only some familiar keywords are selected for training a classifier, the value of recall is not high. Additionally, the scale of the experiments data needs to be extended. Due to the difficulty of getting the truth of the type of the app reviews by manually labeling, we only select 1924 reviews for the experiments. It inevitably limits the verification experiments on the performance of the classification algorithms.

4 Related Work

Many works have been presented to extract feature requests. Laurent et al. [5] explore the use of online forums to conduct the requirements engineering tasks of the open source projects which was led by the software vendor. Castro-Herrera et al. [6] present a hybrid recommender system to identify potential users who might be capable of responding to unanswered posts in open source forums. Castro-Herrera et al. [7] utilize the organizer and promoter of collaborative ideas (OPCI) recommendation system to recommend expert stakeholders of the field and the requirements are elicited by means of the domain knowledge from these expert stakeholders. These approaches focus on the problem of finding the proper stakeholders to participate in the process of requirements elicitation. Hariri et al. [8] and Dumitru et al. [9] leverage the data mining techniques to extract the common features from online products description and design a recommender system to elicit missing features.

App reviews have also been used to extract feature requests. Popescu et al. [10] introduce an unsupervised information extraction system named OPINE to build a model of important product features. Galvis Carreo et al. [11] adapt topic modeling from user comments to extract the topics mentioned and some sentences representative of those topics. These approaches do not consider sentiment of the user and they do not address the problem of the mis-classification or mix of topics. Iacob et al. [12] design a prototype named mobile app review analyzer (MARA) to automatic retrieve mobile app feature requests from app reviews, where they manually define linguistic rules and identify feature requests from reviews which match at least one linguistic rule. Because only part of the linguistic rules are listed in their paper, it is difficult to quantitatively compare their approach with ours. Their approach needs much manual labors and the output of their approach is the corresponding keywords relevant to the feature requests. In our approach, we use linguistic rules and bag of words as classification attributes to train a classifier for identifying reviews on feature requests. We also use the Stanford Parser to extract phrases to represent feature requests, which can be easily understood by users. Guzman et al. [13] propose an automated approach that helps developers filter, aggregate, and analyze user reviews. But they mainly focus on the emotion analysis on reviews and feature requests mining is not their research task. Maalej et al. [14] introduce several probabilistic techniques such as string matching, text classification, NLP (Natural Language Processing) and sentiment analysis, and compare the classification algorithms including Naive Bayes, Decision Tree and maximum entropy (MaxEnt) to classify app reviews into four types: bug reports, feature requests, user experiences, and ratings. They classify app reviews comprehensively, but they do not consider the characteristics such as linguistic rules only for feature requests. In contrast to their works, we add linguistic rules besides bag of words and metadata as classification attributes to identify the reviews on feature requests. Furthermore, we adopt the Stanford Parser to extract representative phrases as feature requests based on clustered topics and top relevant terms of each topic.

5 Conclusion and Future Work

An approach of extracting feature requests from app reviews is proposed in this paper. The approach can be applied in the early stage of software requirements engineering, which can be a supplement for mining features from the description of mobile apps. In order to accurately extract feature requests from reviews, it is important to identify whether a review belongs to feature requests. Therefore, different classification algorithms are compared under various classification attributes, which are validated by leveraging a real world data set of reviews from the AppleStore and the Google Play stores. In addition, phrases that represent feature requests are extracted by using word dependencies based on the clustered reviews.

In the future, we plan to extend our work from the following directions. Firstly, we will further investigate how to extract more meaningful phrases as feature requests from reviews. Secondly, we plan to evaluate the performance of various classification algorithms under different classification attributes when the experimental data set grows to a larger scale.

Acknowledgments. The work is supported by the National Basic Research Program of China under grant No. 2014CB340404, and the National Key Research and Development Program of China under grant No. 2016YFB0800400, and the National Natural Science Foundation of China under Nos. 61672387, 61373037, 61572186 and 61562073. The authors would like to thank anonymous reviewers for their valuable suggestions.

References

1. Tiwari, S., Rathore, S.S., Gupta, A.: Selecting requirement elicitation techniques for software projects. In: the CSI 6th IEEE International Conference on Software Engineering (CONSEG), pp. 1–10. IEEE Press, New York (2012)
2. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie Mellon University (1990)
3. Radatz, J., Geraci, A., Katki, F.: IEEE Standard Glossary of Software Engineering Terminology. IEEE Std **610121990**(121990): 3 (1990)
4. Cleland-Huang, J., Dumitru, H., Duan, C., Castro-Herrera, C.: Automated support for managing feature requests in open forums. Commun. ACM **52**(10), 68–74 (2009)
5. Laurent, P., Cleland-Huang, J.: Lessons learned from open source projects for facilitating online requirements processes. In: Glinz, M., Heymans, P. (eds.) REFSQ 2009. LNCS, vol. 5512, pp. 240–255. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02050-6_21
6. Castro-Herrera, C.: A hybrid recommender system for finding relevant users in open source forums. In: 3rd IEEE International Workshop on Managing Requirements Knowledge, pp. 41–50. IEEE Press, New York (2010)
7. Castro-Herrera, C., Cleland-Huang, J., Mobasher, B.: Enhancing stakeholder profiles to improve recommendations in online requirements elicitation. In: Proceedings of the 17th IEEE International Conference on Requirements Engineering, pp. 37–46. IEEE Press, New York (2009)

8. Hariri, N., Castro-Herrera, H., Mirakhorli, M., Cleland-Huang, J.: Supporting domain analysis through mining and recommending features from online product listings. IEEE Trans. Softw. Eng. **39**(12), 1736–1752 (2013)
9. Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., Mirakhorli, M.: On-demand feature recommendations derived from mining public product descriptions. In: Proceedings of the 33rd IEEE International Conference on Software Engineering, pp. 181–190. IEEE Press, New York (2011)
10. Popescu, A.M., Etzioni, O.: Extracting product features and opinions from reviews. In: Anne, K., Stephen, R. (eds.) Natural Language Processing and Text Mining, pp. 9–28. Springer, London (2007)
11. Galvis Carreño, L.V., Winbladh, K.: Analysis of user comments: an approach for software requirements evolution. In: Proceedings of the 35th IEEE International Conference on Software Engineering, pp. 582–591. IEEE Press, New York (2013)
12. Iacob, C., Harrison, R.: Retrieving and analyzing mobile apps feature requests from online reviews. In: Proceedings of the 10th IEEE Working Conference on Mining Software Repositories (MSR 2013), pp. 41–44. IEEE Press, New York (2013)
13. Guzman, E., Maalej, W.: How do users like this feature? A fine grained sentiment analysis of App. reviews. In: Proceedings of the 22nd IEEE International Conference on Requirements Engineering, pp. 153–162. IEEE Press, New York (2014)
14. Maalej, W., Nabil, H.: Bug report, feature request, or simply praise? On automatically classifying app reviews. In: Proceedings of the 23rd IEEE International Conference on Requirements Engineering, pp. 116–125. IEEE Press, New York (2015)
15. Torgo, L.: Data Mining with R: Learning with Case Studies. Chapman & Hall/CRC, Boca Raton (2010)
16. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. (TIST) **2**(3), 271–2727 (2011)
17. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
18. Chen, D., Manning, C.D.: A fast and accurate dependency parser using neural networks. In: 2014 Conference on Empirical Methods in Natural Language Processing, pp. 740–750 (2014)
19. Wu, J., Chen, L., Zheng, Z., Lyu, M., Wu, Z.: Clustering web services to facilitate service discovery. Knowl. Inf. Syst. **38**(1), 207–229 (2014)
20. Chen, L., Wang, Y., Yu, Q., Zheng, Z., Wu, J.: WT-LDA: user tagging augmented LDA for web service clustering. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 162–176. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45005-1_12
21. Pagano D., Maalej, W.: User feedback in the appstore: an empirical study. In: Proceedings of the 21st International Conference on Requirements Engineering, pp. 125–134. IEEE Press, New York (2013)
22. Thelwall, M., Buchley, K., Paltoglou, G.: Sentiment strength detection for the social web. J. Am. Soc. Inf. Sci. Technol. **63**(1), 163–173 (2012)
23. Wang, J., Zhang, N., Zeng, C., Li, Z., He, K.: Towards services discovery based on service goal extraction and recommendation. In: 2013 IEEE International Conference on Services Computing, pp. 65–72. IEEE Press, New York (2013)