Received: 20 December 2020 | Accepted: 30 July 2021

IET Software

DOI: 10.1049/sfw2 12042

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH PAPER

Mining detailed information from the description for App functions comparison

Huaxiao Liu[1,2] | Xinglong Yin[1,2] | Shanshan Song[1] | Shanquan Gao[1,2] | Mengxi Zhang[1,2]

[1]College of Computer Science and Technology, Jilin University, Changchun, Jilin Province, China

[2]Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun, Jilin Province, China

Correspondence

Shanshan Song, College of Computer Science and Technology, Jilin University, Changchun, Jilin Province 130012, People's Republic of China.
Email: songss@jlu.edu.cn

Funding information

Interdisciplinary Research Funding Program for Doctoral Students of Jilin University, Grant/Award Number: 101832020DJX064; Graduate Innovation Fund of Jilin University, Grant/Award Number: 101832020CX181; Natural Science Research Foundation of Jilin Province of China, Grant/Award Number: 20190201193JC; National Key Research and Development Program of China, Grant/Award Number: 2017YFB1003103

Abstract

The rapid development of Apps not only brings huge economic benefit but also causes increasingly fierce competition. In such a situation, developers are required to develop and update innovative functions to attract and retain users. Afterwards, analysing the functions of similar products can help developers formulate a well-designed plan at the beginning of development as well as make updated strategies during the version update process. However, although there have been some methods that can be applied to extract the features from App descriptions to achieve this purpose to some extent, the features they obtained do not cover the details of App functions. Therefore, to conduct an in-depth research on App functions, a novel method is proposed to extract App features with detailed information and an approach to integrate the gained results for further helping developers obtain the valuable knowledge better is provided. Subsequently, a series of experiments is carried out to evaluate our method. The results reveal that the proposed method can mine the features with detailed information from descriptions and integrate them effectively and also can assist developers to compare with other competitors and develop a better competitive analysis scheme.

1 | INTRODUCTION

The emergence of smartphones marks our entry into a new stage of mobile Internet, and mobile applications (Apps) that come with it have become an indispensable part of our life. The statistics have reported that there are more than 3.5 billion smartphone users all over the world, and the number of downloaded Apps is more than 200 billion from 2018 to 2019. At the same time, until the third quarter of 2020, the number of available Apps hosted on Google Play reached 3.04 million [1], and there are still more Apps launching in the market constantly. With such numerous Apps, the number of Apps with similar functions will also increase continually, and they will satisfy similar user requirements and share a common user group. Thus, these Apps will generate a fiercely competitive relationship with each other [2, 3]. Take the APKPure as an example, which is a website like Google Play that allows users to download '.apk' files. There are 305 android Apps in the category of SOCIAL, and 51 Apps out of them can be gained by searching with 'video chat' as keywords.

To develop a successful App amidst such fierce competition, both in the early stage of App development and in the process of version update, developers need to compare similar Apps to understand their strengths and weaknesses [4], analyse the reasons why Apps have different levels of popularity, and further adjust proper marketing strategies and purposes for their Apps. In this case, the Apps they develop will be more attractive to users and can be distinguished from others. To achieve the above goal, the developers are not only required to determine their target customers and mine the requirements

but also need to compare the functions of different Apps. Generally, the developers try to read App descriptions in the same domain, or even download some of them to understand the differences among these Apps. If there are few Apps that need to be compared, then it is easy for developers to complete it manually. But with the number of Apps growing explosively nowadays, it is hard to compare all Apps within a limited time.

The App description is designed to assist users to have a preliminary understanding of the App, and it contains plenty of features about the App functions. Such features consist of two parts: one aims to directly describe the subject of functions and indicates what the Apps can do, while the other gives some detailed information about the functions and it explains how the Apps complete from the perspective of implementation, scope, privacy protection, and so on. Figure 1 shows the description of Instagram, one is the contents marked with yellow that are the subject of functions, and the other is the content marked with blue that are the details of these functions. For example, the second sentence introduces that the subject of functions is 'browse photos and videos' and explains the details of this function that you can use Instagram to browse photos and videos 'from people you follow in your feed'.

Some existing tools (CLAP [5], MSR [6], and SAFE [7] etc.) are proposed to mine App descriptions and obtain information about App function by extracting App features. However, the features extracted by these methods are mainly composed of the subject of functions, such as verb–noun pairs, consecutive words, and word combinations, but they all ignore the detailed information of the App function. Please note that in Table 1, the first column lists some sentences collected from App descriptions under the SOCIAL category in Google Play, and then we adopt some existing methods to extract features from these sentences and the results obtained are listed in the second column. It reveals that many Apps have the function of 'add friend', but how they implement the function of 'add friend' is unknown, such as 'on Instagram', 'by scanning their nametags', or 'from different countries'. Although the information in the second column can be used to mine domain knowledge, it is too general and highly abstract. Relative to this information, developers are more inclined to know the details of the functions, shown in the third column. In fact, the detailed information can help developers locate the differentiated requirements of users in the current situation of App market segmentation, with the aim of helping developers come up with a more unique plan for developing and updating the Apps. Meanwhile, assisting developers in comparing with other competitors and developing better competitive analysis schemes is another benefit of it.

In terms of most feature extraction methods, it is hard to cover the details of App functions. We propose a novel method to extract App features with detailed information, and then we utilise these features to help developers mine more valuable information. Notably, Figure 2 denotes the overview of our

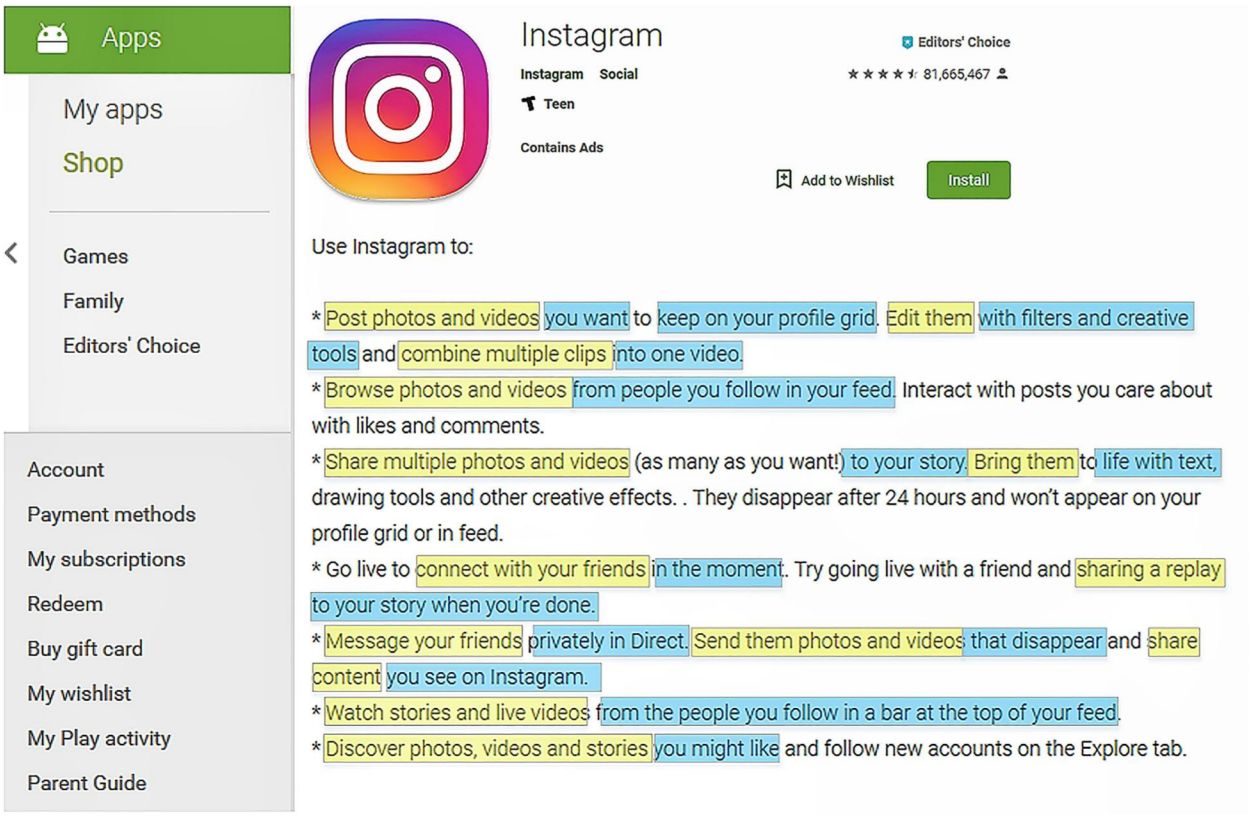

FIGURE 1 An instance of descriptions where functions and detailed information are marked

TABLE 1 The subject and detailed information of functions

Sentence	Subject of function	Detail of function
Quickly add friends on Instagram by scanning their nametags. (Instagram)		On Instagram, by scanning their nametags
Easily add all your friends from Facebook, Twitter and the address book. (Frontback)	Add friends	From Facebook, Twitter and address book
Add people in a group from different countries. (Lucky chat)	Add people	In a group, from different countries
Add new streak friends on the Messenger App by usernames search. (KK friends)		On Messenger App, by usernames search
Find people who share your interests. (MeetMe)		who share your interests
Find friends who share the same passion with you! (BOO)	Find people	Share the same passion with you
You can find people in your neighbourhood using a radar. (VeroMeet)	Find friends	In your neighbourhood
You can create your own event to find people interested near you. (Meeve)		Interested near you
Send a private message to individuals easily. (Pinoy Australia Forum)		Private message, to individuals easily
Send a message via WhatsApp. (Birthday Anniversary reminder)	Send message	via WhatsApp
The person you sent the message to will never know who you are. (xSay2)		Never know who you are
Send a message to any WhatsApp number without saving the number. (Status saver)		To any WhatsApp number, without saving the number

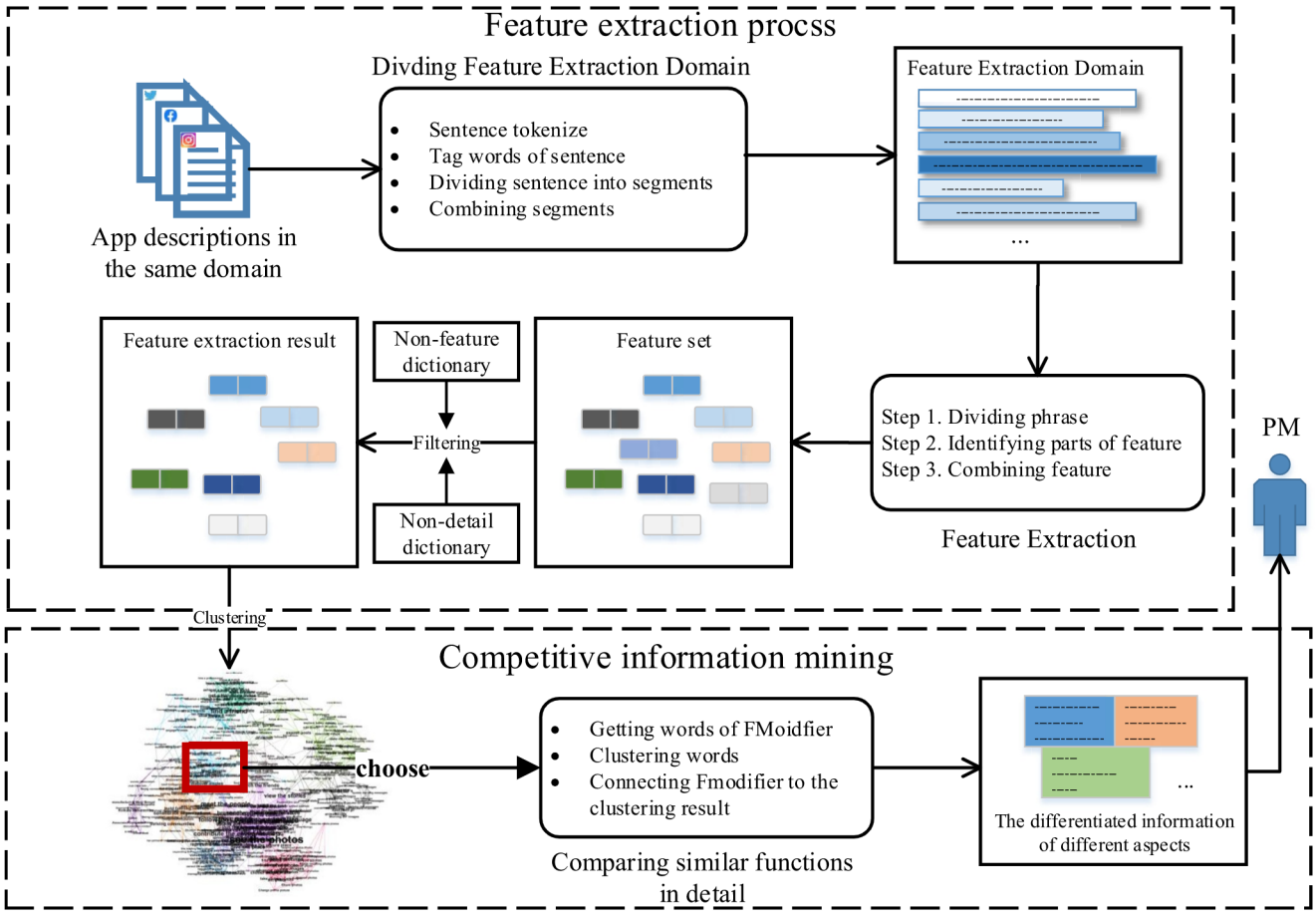

FIGURE 2 The overview of our method

proposed method, which is composed of two main parts. In the first part, we first give a new feature definition that contains detailed information of App functions, second, we divide the scope of feature extraction in the descriptions, third, we identify and combine features, and finally, we filter out the results that cannot describe the functions of Apps. The second part aims to collect competitive information to mine the differentiation of functions. We cluster the features describing similar App functions and then divide the detailed information in a cluster into different aspects. With such a method, developers can obtain the features that can accurately describe App functions in an automated way and observe the comparison results between the implementation forms of the App functions.

With regard to the method of extracting detailed information, we have implemented and applied it to 41,345 Apps on Google Play under the categories of SOCIAL, EDUCATION and PHOTOGRAPHY, and we have also thoroughly evaluated each step of our method. First, we carry out an experiment to evaluate the performance of our method in extracting features, achieving the F-measure of 71.6% on average. The second validation stage aims at discussing whether our method can cluster similar App functions and compare it with manual clustering results so that we can cluster similar features together. Last but not the least, we provide the results of this method to 20 developers and receive their feedback about the applicability of it. The feedback denotes that the information obtained by our method can be well utilised to encourage them to get inspiration.

In summary, this study makes the following contributions:

1. We implement a new feature extraction method, which can extract the detailed information of App functions from App descriptions.
2. We compare the App functions and mine the corresponding distinguished information, which can lay the foundation for differentiated user requirements analysis in the market segmentation.
3. We evaluate each step of our method and distribute our result to 20 developers, and the experimental results and the feedback of the developers reveal that our method is effective in comparing App functions.

The remainder of this study is organised as follows: Section 2 gives a formal definition of the feature and presents our feature extraction method in detail. In section 3, we introduce the process of mining competitive information from features. Section 4 presents our evaluation results and shows the feedback of 20 developers on our method. Section 5 introduces studies that are related to this study. Finally, the conclusion and future work are shown in section 6.

2 | FEATURE EXTRACTION METHOD

Developers can extract features with a variety of methods when they analyse App functions from the related documents, such as performing a word frequency and co-location analysis to find the words that associate frequently [6], post-tagging the text to extract verb–noun pairs [8, 9], and using part-of-speech patterns [7]. Although these methods may extract useful information for App domain analysis, overall evaluation for the App, feedback analysis of users, and so on, they cannot support the in-depth comparison of App functions.

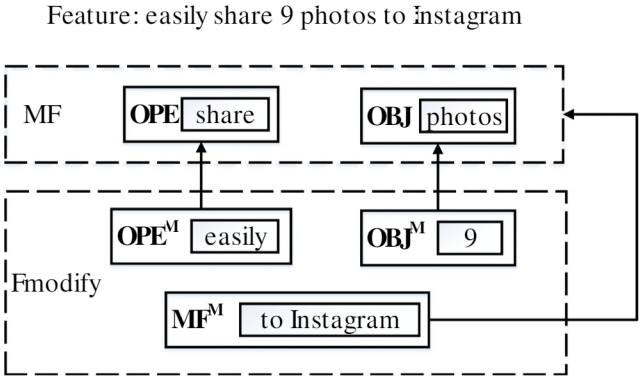

FIGURE 3 An example of a feature and the content it contains

Thus, we give a formal definition of the feature that contains the details of App functions and extract features with the following three steps: First, the sentences of App descriptions are divided into segments called Feature Extraction Domains to reduce the range of feature extraction. Second, the words in the feature extraction domain are divided into different phrases; some verb phrases and noun phrases are combined as the subject of the feature according to specific rules, and the phrases describing the details of functions are chosen as the modifier of a feature. Third, we have built two dictionaries to filter out the part of the feature that cannot describe the function and its details.

2.1 | Formal definition

Feature ***F*** is a list of words that describe the App function and details of the function, F is defined as a two-tuple (MF, FModifier). Among them, **MF** refers to the **main subject of feature**, which is composed of a verb and a noun. **FModifier** is **the modifier of feature**, which is the words appearing after the MF in the feature to add some detailed information about the App function. Specifically speaking, the **MF** is defined as a two tuple (OPE, OBJ), OPE describes the action of App and OBJ describes the entity of action. **FModifier** is defined as a set of word lists, FModifier = $OPE^M \cup OBJ^M \cup MF^M$, and OPE^M, OBJ^M, MF^M are supplements to OPE, OBJ and MF.

Taking Figure 3 as an example, 'easily share 9 photos to Instagram' is a F. In this feature, 'share' is an OPE, 'photos' is an OBJ. 'share photos' is a MF, 'easily' is an OPE^M, '9' is an OBJ^M and 'to Instagram' is a MF^M.All of them reflect what functions the App can support and how to achieve them.

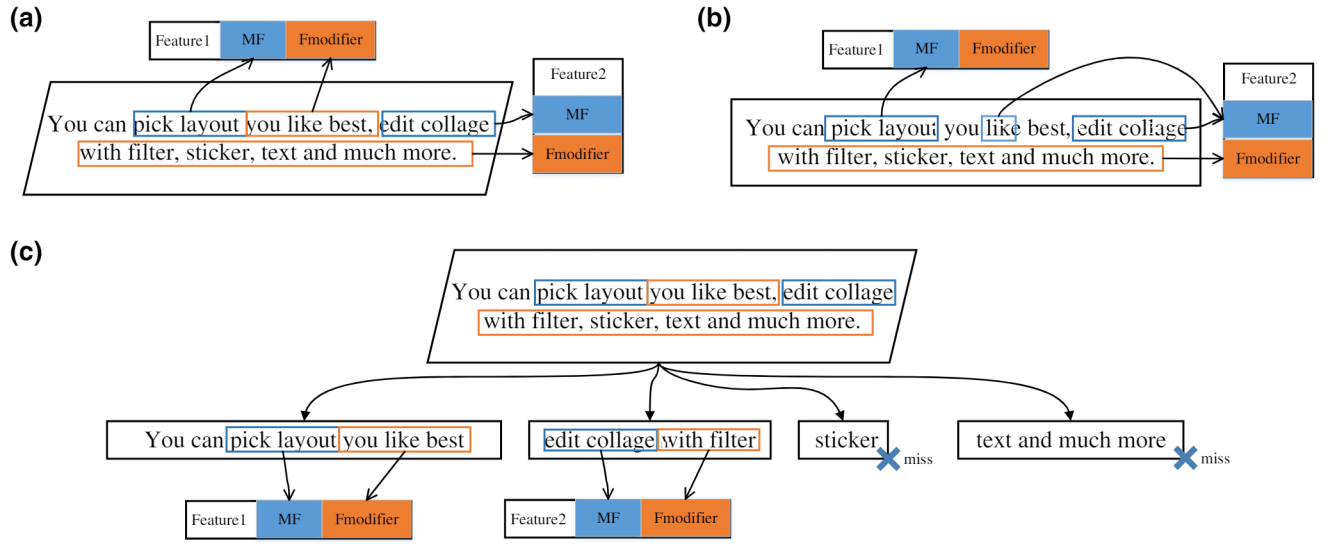

FIGURE 4 An example of extracting features in different FED

2.2 | Dividing feature extraction domain

Feature Extraction Domain (FED) refers to the unit of searching the words that belong to a feature during the process of feature extraction. To distinguish the FED, existing methods usually use punctuations to divide the text into natural sentences or clauses, but it is not enough to support Fmodifier extraction. As shown in Figure 4, we take this sentence, '*You can pick layout you like best, edit collage with filter, sticker, text, and much more*', as an example to extract features in the FED. There are two functions that are mentioned in this sentence, and the reasonable feature extraction results are shown in Figure 4(a). However, as shown in Figure 4(b), if we regard the whole sentence as a FED, the feature will be composed of the word 'like' and 'edit collage', which may lead to the error condition. Except when using punctuations to make a division, if extracting feature in clauses dividing by comma, then 'edit collage with filter, sticker, text, and much more.' will be divided into three FEDs, and the modifiers of 'edit collage' will no longer be complete, see Figure 4(c).

Therefore, we propose a method to divide FEDs and make sure that there is at most one feature in a FED, which follows the next four steps:

1. Divide the App descriptions into some sentences.
2. Tag all words in each sentence by the POS-tag of the Natural Language Tool Kit (NLTK)[10].
3. Divide each sentence into shorter fragments by punctuations, conjunctions (such as and, or) and some prepositions, such as 'to' and 'for'.
4. Combine these fragments in a FED based on three rules.
   (a) If the fragment has both the verb and noun in sequence or the noun and passive voice in sequence, this fragment must be in a FED so that this fragment can be regarded as the start of a new FED;
   (b) If the fragment contains only verbs, it will be merged with the following fragment into a FED;
   (c) Apart from the above two rules, we will add this fragment to the previous FED.

There are some examples of partitioning the FED in Figure 5 (a) to (b). The examples in Figure 5 reveal that these sentences can be divided into fragments, which are contained in blocks with different colours. The fragment that conforms to rule (a) is contained in the black blocks, rule (b) is contained in the white blocks, and rule (c) in the green blocks. After that, we connect each fragment based on corresponding rules and construct the FEDS, see Figure 5 (b).

2.3 | Feature extraction process

After defining the features and dividing the FED, in this section, we will explore how to extract features in the FED. First, we need to identify the composition in the FED such as OPE, OBJ, OBJ^M, then combine these compositions according to the rules in Table 2. But we note that there are special cases where some OBJs and MFs should be adjusted to FModifiers. Subsequently, we will introduce each step in detail from two perspectives:

***General Cases*:** The general cases include the following three conditions:

Dividing phrase. According to the POS-tag of words, the FED can be divided into several phrases. In this step, continuous verbs are regarded as verb phrases, and continuous nouns are regarded as noun phrases. The linking verb 'be' is regarded as a single phrase, prepositions are treated as the beginning of modified phrases, and the remaining words are directly used as modified phrases.

Identifying OPE, OBJ, OPE^M, and OBJ^M. A verb phrase can be regarded as an OPE directly, and the modified phrase

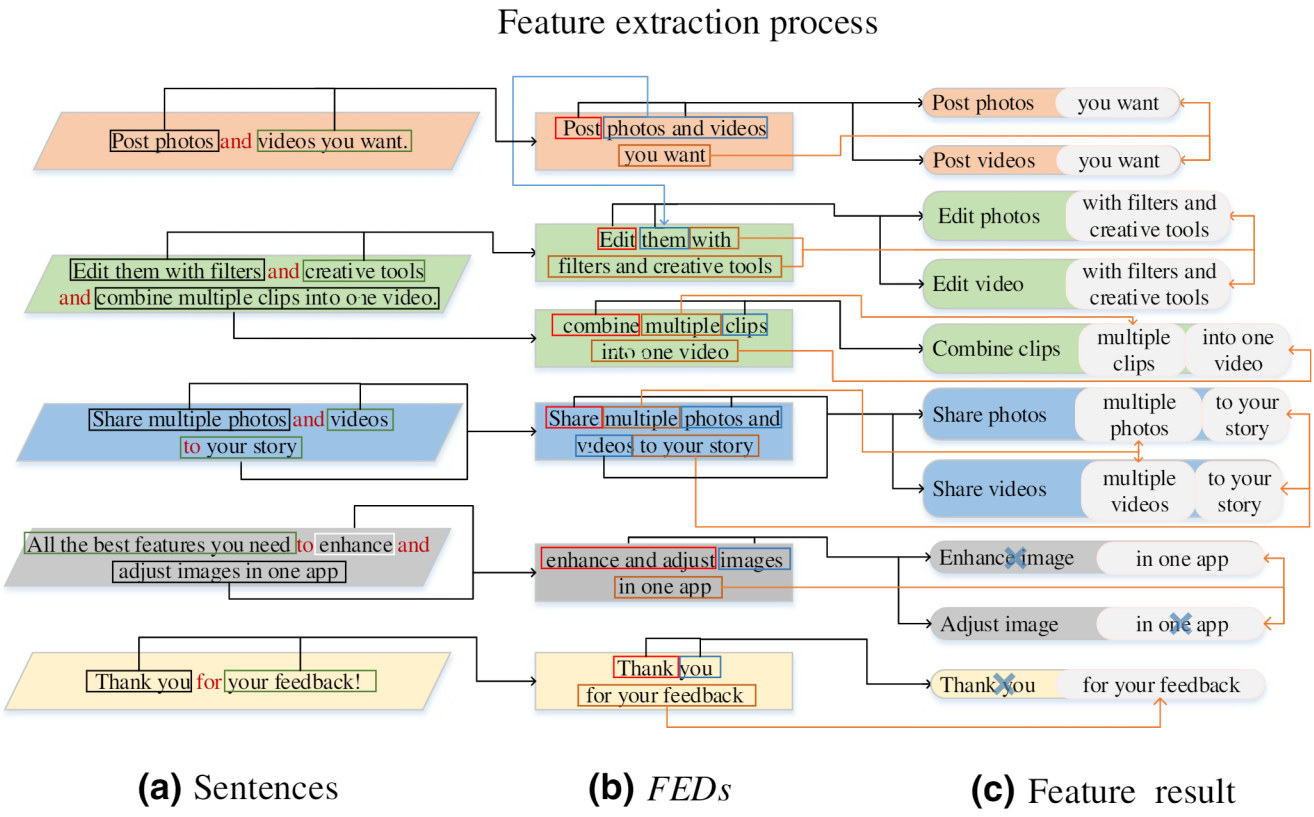

FIGURE 5 The process of feature extraction

TABLE 2 The rules for feature extraction

Rules	Explanation	Example
OPE OBJ	In this case, the Apps will first propose that the software can perform the operation, after which the operation will follow the object that the software operates.	Send one-to-one private messages
OBJ BE OPE	In the passive case, the object will appear first in the function description and then the operation	Volume control slider was added for in-app sound effects.
BE OBJ	Same as the first rule, the description usually uses BE verbs to describe the App function, but we do not extract the BE verb in the result.	Collage is the best photo collage maker.

appearing in front of the OPE can be used as OPE^M. More generally, a noun phrase can be regarded as an OBJ and a modified phrase appearing in front of an object belongs to OBJ^M.

Identifying MF, MF^M**.** OPE and OBJ are grouped into candidates of the MF regardless of the modified phrase in them, according to the rules in Table 2. Under most conditions, the candidate of the MF is the MF, and all phrases from the MF to the end of the FED belong to the MF^M.

***Special Cases*:** The special cases involve the following three situations:

Extending FModifier. In certain cases, an OBJ may modify another OBJ or a candidate for MF may modify another MF. For instance, in the sentence of 'a large number of photos', 'a large number of' is the modification of the OBJ 'photos'. Please note that another sentence 'Support 15 photos for making collage', 'support 15 photos' is the modification of the MF 'making collage'. Besides, on account of those cases usually accompany the appearance of prepositions, such as 'for', 'of', 'to', and 'by', so we judge the relationship in the OBJs or candidates for MF according to those prepositions, and then append these OBJs and the candidate of MF to FModifier.

Replacing pronoun nouns. Regarding the pronoun in the text (such as 'it', 'they' and 'them'), we choose the OBJ appearing in the previous sentence to replace the pronoun. Considering this special case helps to prevent pronouns from appearing in features that may cause confusion. Besides, after observing and analysing large-scale features with pronoun nouns, we find that 76% of them can be well replaced.

Merging the parallel OPEs and OBJs. We treat the OPEs or OBJs as an OPE or OBJ temporarily when there are conjunctions connecting multiple OPEs or OBJs. After all the features are well extracted, they are split into multiple features, and these features share the same FModifier. For example, in the process of extracting features from 'share photos and videos with friends', the 'photos and videos' can be regarded as one OBJ to identify the MF according to the feature extraction rules. After identifying the MF, we split it into two features and add their FModifier, 'share photos with friends' and 'share videos with friends'.

Further, some features that cannot describe App functions or details are incorrectly extracted, such as 'fix bug', 'improve experience', 'quickly', 'you like' etc. These features have a similar syntax structure to the functional description but are unhelpful when comparing Apps. To filter them out from feature extraction results, we summarise the frequent words appearing in them and construct two dictionaries, a non-function dictionary and a non-detail dictionary. If a MF or FModifier consists mainly of words in a dictionary, we will remove this feature or this FModifier. Notably, apart from the above special cases, there may be other situations that we cannot predict currently; but fortunately, our proposed method has high scalability and can adapt the new situations by extending the rules.

There are some examples to explain how features can be extracted from the FED, see Figure 5 (b) to Figure 5 (c). Focussing on each FED, the words in the red block refer to OPE, the words in the blue block are OBJ, and the orange block is FModifier. In this process, 'them' in the third FED is replaced with 'photos and videos', and then they are divided into two features because of parallel OBJs. As a result, we remove some features and FModifiers according to our filtering method.

3 | COMPETITIVE INFORMATION MINING

Developers are inclined to compare the functions they are concerned with or some important information when they perform development or update. For each comparable App in a particular domain, we extract the features containing modifications from their description texts by utilising the method mentioned in Section 2. However, it is hard for developers to read and digest the comparison information if all the information about the obtained features is only directly listed. To assist developers in learning from other products better, we propose a method to cluster features as well as FModifiers to further show a comprehensible analysis result, and we will explain this method in a detailed manner in this section.

3.1 | Measure features similarity

In general, similar features often have a similar MF. So we can calculate the similarity between features by comparing their MF. However, the OPE and OBJ in the MF normally contain 1 to 5 words, resulting in errors if we only calculate the average similarity of each word or directly adopt the most similar word to evaluate the similarity of a phrase. Notably, this situation is mainly caused by the number of words between different phrases.

To calculate the similarity of the phrases, we adopt the Word2Vec to vectorise the words of descriptions in each App. We then evaluate the similarity between words by calculating the cosine similarity of their vectors. Further, to examine the threshold utilised to determine whether the words are similar, we manually analyse a large amount of data, and find two possible values: 0.2 and 0.7, as for how to determine these two values, see Section 4.1.3. As shown in Table 3, the correlation between two words is weak if their similarity is less than 0.2, and the correlation between two words is strong if the similarity is larger than 0.7. Thus, we (i) deem that words are similar if their similarity is larger than 0.7; (ii) treat words as unrelated when their similarity is less than 0.2; and (iii) think that the similarity between two words is fuzzy if their similarity is within the range of [0.2, 0.7].

By manually observing existing phrases, we discover that (i) two phrases are usually not similar if the most similar word pair between them is not similar, while (ii) two phrases are similar if the most similar word pair between them is similar, and there is no obvious inconsistency in the meaning of remaining words in two phrases. Based on this observation, we give an iterative method to calculate the similarity among all phrases: first, the similarity between the most similar word pair in two phrases is regarded as the initial value of the two phrases. Second, we search out the most similar word from the remaining words of two phrases in turn and then adjust the similarity between phrases by utilising the formula 1 and remove the word pair from these phrases.

$$Sim = \begin{cases} c \cdot Sim & s \le A \\ Sim & A < s < B \\ 1 - c \cdot (1 - Sim) & s \ge B \end{cases} \quad (1)$$

In formula 1, where Sim refers to the similarity value between two phrases, s is the similarity between the selected word pair and c denotes the corrected parameter (0.7 in our experiments). The experimental results support us to determine A = 0.2 and B = 0.7 in this formula; more detailed explanation can be found in Section 4.1.3. In addition, this process needs to set a specific number of iterations to determine where it stops extracting, which in our experiments is set to 5.

By utilising the above method, we can calculate the similarity of OPE and OBJ in features and adopt DBSCAN [11] to cluster features based on these results. In such a cluster method, two features will be considered as similar features only if a part of them is similar at the same time. Further, please note that the thresholds utilised to determine whether OPE and OBJ are similar or different are OPE: Sim ≥ 0.5 and OBJ: Sim ≥ 0.6 in our experiments.

TABLE 3 The types of word pairs

Unrelated word pairs			Fuzzy word pairs			Similar word pairs		
Content	year	0.0398	Photo	Poster	0.6481	Photo	Picture	0.7272
Represent	Get	0.0008	Share	Send	0.539	Discover	Explore	0.7451
Share	Change	0.1992	Gift	Romance	0.5284	Girlfriend	wift	0.7886
Others	Keyboard	0.0152	Get	Make	0.4203	Change	Adjust	0.7061

3.2 | Analyse the clustering result of similar features

The FModifiers describe the specific implementation method adopted by Apps to implement similar functions. However, as the number of Apps with similar features gradually increases, the FModifiers are usually still varied even in the same cluster. Thus, to help developers understand this information better, we further divide the information under the same cluster into different aspects based on its content.

We have considered using the algorithm mentioned in section 3.1 to calculate the similarity between FModifiers to further complete the clustering work, but the data in each cluster gained by utilising this method is almost identical, and this deviates from our purpose of clustering the FModifiers of the same aspect together. Hereby, after analysing the practical data, we find that the way to implement App functions is often determined by some information in the FModifier, such as a sentence of 'Send a private message to individuals easily', its corresponding FModifier is 'private message', which reflects the purpose that developers are required to process primarily is 'private message'. This example means that if the FModifier contains the same or similar words for two features, the details they described will also be similar. Based on the obtained findings, we propose an approach to cluster the FModifiers in the features. First, we use all words in the FModifiers to build a dataset and remove the words without information from this set by using our list of stop words. Second, we cluster these features via their vectorised representation [12]. Third, we associate FModifiers with the word clusters by probing whether the clusters contain the words in FModifiers. At this point, the FModifiers in features can be divided into different aspects. Table 4 displays the clustering result of FModifiers in the feature cluster in which the centre node is 'share photo', and it can be seen from Table 4 that our method can indeed divide the FModifiers into different aspects according to the contents described by them (Table 5).

Figure 6 shows an example of the information provided by our method, and it can well assist us in illustrating how our method helps developers. Firstly, the developer selects a category and uses our method to extract features and the detailed information from descriptions in this category of Apps. Then, we cluster the features based on the method for calculating similarity, and it shows the centre nodes of feature clusters by utilising gephi so that developers can understand the main functions in the category selected by the developer. Finally, the developer selects a cluster that s/he is interested in, and we divide the detailed information under this cluster into different aspects, then show the results to the developers and help them further understand how different Apps achieve the same function.

4 | EXPERIMENT

We evaluate our method from two dimensions: effectiveness and usefulness, and we conduct a series of experiments to answer the following research questions:

- **Question 1 (effectiveness):** Whether our approach can gain competitive information from App descriptions effectively?
- **Question 2 (usefulness):** Whether the information obtained by our method is useful for developers when they compare similar Apps?

To answer the above questions, we select 3 categories of Apps on Google Play as the subjects of our experiment, including 'Social', 'Photograph' and 'Education'. There are two main reasons for selecting these categories as subjects: one is that the features of these categories are quite different from each other; so we believe that using them as experimental subjects can validate the generalisation of our method. The other is that there are a huge number of Apps in them; so the dataset can cover different kinds of practice situations better.

As shown in Table 5, we select 41,345 Apps from three categories to establish the dataset. For each App, we also collect the description text and the rating by using our crawler from Google Play. Then, we perform statistics on the features extracted from the App descriptions. On average, about 20 features can be extracted from each App description, and about 20% of the features have detailed information of the App function. It indicates that the App description contains a sufficient number of App function descriptions and functional details to support the comparison of similar App functions.

The participants in our experiments included three students and 20 developers; detailed information is given in Table 6.

1. 3 students in our experiments have a Ph.D. degree from Jilin University in China, and they have majored in software engineering. Two of them have 1 to 3 years of industrial experience in software development, and all of them have certain background knowledge in the research of Apps.

TABLE 4 The aspects of the *FModifier*

Key words	*FModifier*
Family colleague classmate	With your friends and family members
	With your classmates
	With your colleagues
	With friends and family
Image photo pics picture	From your phone camera to the photo library
	In a simple photo stream
	Meaningful image quotes
	Your favourite pet pics
	The Anniversary picture
Media network platform	On WhatsApp, Facebook and other social media platforms directly
	To different social media
	Of the social network
	via Line, Facebook, Twitter, Google+, WhatsApp, E-Mail, printer and many more

TABLE 5 Overview of data collection

	Social	Photography	Education
The number of App	5,223	12,296	13,826
The number of features	76,021	301,770	330,928
Average number of App features	14.6	24.5	13.9
Feature number with the FModifier	15,413	55,201	70,401
Proportion of the FModifier	22.73%	19%	23.53%

These doctors are responsible for labelling the samples in the truth sets of Question 1.

2. 20 developers are from three companies, and all of them have more than 2 years of software development experience. We require developers to evaluate the usefulness of the information provided by our method.

4.1 | The experiment for question 1

To evaluate the effectiveness of our method in Question 1, we further design 2 sub-questions to provide an explanation:

- **sub-question (1):** Whether our method can extract features from App descriptions effectively?
- **sub-question (2):** Whether our feature similarity method can cluster similar App functions together?

4.2 | Experimental design for sub-question (1)

To answer the first sub-question, we randomly select 10 Apps with more than one million users in each category and then adopt their descriptions as the data basis to evaluate the performance of our method for extracting features. The evaluation process consists of two steps. First, we establish two truth sets for each category: the truth set of the MF and the truth set of the FModifier. Three evaluators read the descriptions separately to identify MFs and FModifiers in them. Only when a MF (or FModifier) is identified by all evaluators can it be used as the samples in truth sets directly. For the MF (or FModifier) which is only recognised by one or two evaluators, they discuss together and vote to decide whether to add the MF (or FModifier) to the truth set, noting that all MFs and all FModifiers are used to construct a truth set separately.

Second, we use our method to extract the MFs and all the FModifiers in 10 description texts of the category and compare the results with samples in the truth set to evaluate the performance of MF extraction and FModifier extraction in each category. Here, we give 3 parameters: TP, FP and FN—TP, the number of MFs (FModifiers) contained in both the extraction result and the truth set; FP, the number of MFs (FModifiers) contained in the extraction result but not in the truth set; FN, the number of MFs (FModifiers) contained in the truth set but not extracted by our method. Based on these parameters, we evaluate the result of two kinds of data as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

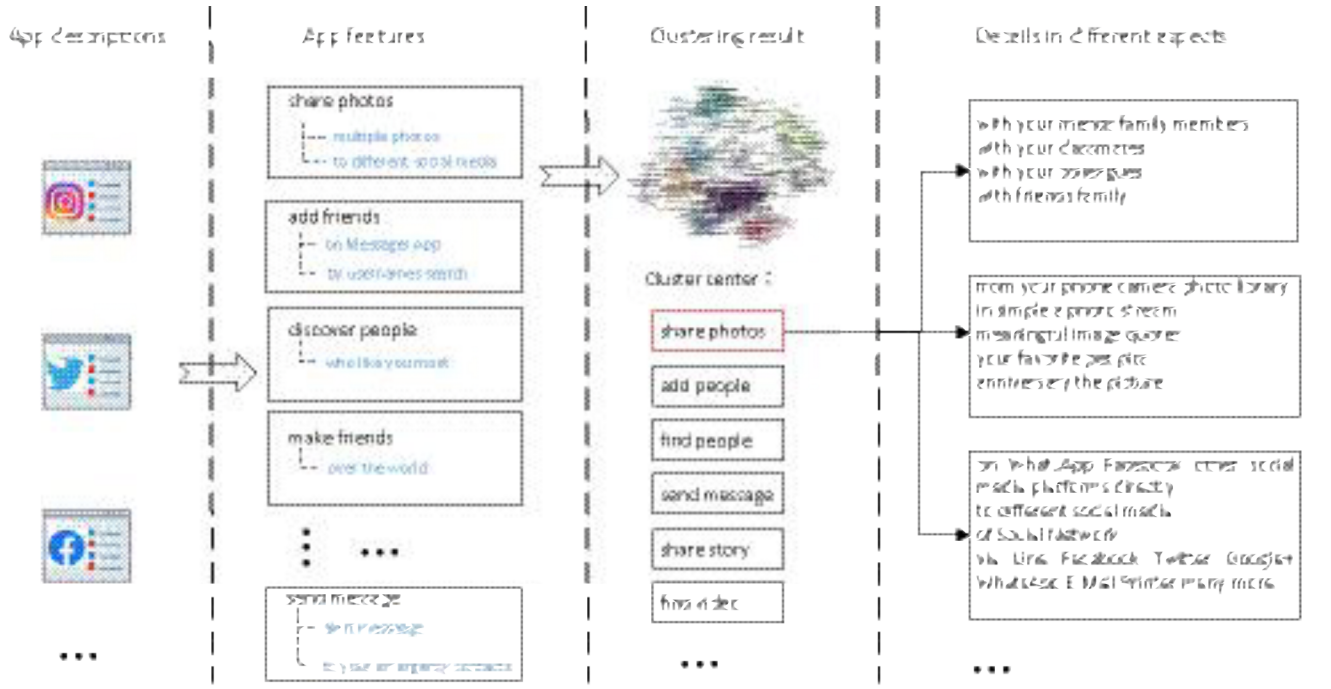

FIGURE 6 The main information that our method provides

TABLE 6 The participants in experiments

	Number	Age	Major	Development skills
Doctors	3	26–31	Software Engineering	2 of them have 1–3 years of industrial experience.
Developers	20	28–36	Software Engineering	All of them have more than 2 years of software development experience.

Besides the above evaluation work, we also carry out an experiment to explore the impact of actions mentioned in Section 2. With regard to extracting the MF, we use the method without considering non-function dictionary filtering and the method without considering both non-function dictionary filtering and extending the FModifier to extract the MF information from descriptions and compare the experimental results with our method for further analysing the impact on the process of extracting the MF. Similarly, we use the method without considering non-detail dictionary filtering and the method without considering both non-detail dictionary filtering and divide the FED to extract the FModifiers in descriptions to explore the impact of these two actions on the process of extracting the FModifier.

4.3 | Experimental result for sub-question (1)

Table 7 shows the experimental result of our method that extracts the feature and detailed information, and we analyse it from two perspectives:

First, it can be seen from Table 7 that our method can extract the MF information in App descriptions effectively: the precision is from 0.6966 to 0.7243, the recall is from 0.6810 to 0.7654, and the average F-measure is 0.7159. In addition, 28.91% of the MFs obtained by our method is wrong. With such wrong results, we observe them and find two main reasons—one is that the POS of many words gained by using the NLTK is not accurate; for example, the word 'share' is often tagged as a noun, while the other is that some useless words cannot be filtered out if our dictionary is used. There are 27.77% of MFs that cannot be extracted by using our extraction rules. We analyse them and summarise two main reasons: firstly, some sentences contain the feature information but they do not meet any extraction rule in Table 2, such as 'frames or grids to choose' in the sentence '100+ Layouts of frames or grids to choose from!'; secondly, the POS of many words gained by using the NLTK is wrong.

Second, the result of Table 7 also indicates that our method can well extract the FModifier information from descriptions: the average precision, recall and F-measure is 0.850, 0.782 and 0.814, respectively. The recall value of our method is high in each category, and this indicates that most of these FModifiers in the descriptions can be gained by our method. However, 15% of the FModifiers that we get by using our method is not in the truth sets, and there are three main reasons according to our analysis: (1) the MFs in some sentences are extracted inaccurately, and this makes their FModifier information wrong; (2) some sentences contain multiple features, and

TABLE 7 The experimental result of our method (MF extraction and FModifier extraction)

	MF extraction			FModifier extraction			
Category	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**	**Proportion**
Social	0.6966	0.7654	0.7294	0.840	0.783	0.810	48.20%
Photography	0.7243	0.7204	0.7223	0.846	0.786	0.815	45.20%
Education	0.7117	0.6810	0.6960	0.865	0.776	0.818	42.30%
Average	0.7109	0.7223	0.7159	0.850	0.782	0.814	45.23%

TABLE 8 The influence of operations on the MF extraction

	Without considering both non-function dictionary filtering and extending the FModifier			Without considering non-detail dictionary filtering		
Category	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**
Social	0.4437	0.7778	0.5650	0.5743	0.7778	0.5806
Photography	0.5153	0.7258	0.6027	0.5633	0.7419	0.6404
Education	0.4550	0.6983	0.5510	0.4655	0.6983	0.5586
Average	0.4713	0.7340	0.5729	0.5344	0.7393	0.5932

TABLE 9 The influence of operations on FModifier extraction

	Without considering both non-function dictionary filtering and dividing FED			Without considering non-detail dictionary filtering		
Category	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**
Social	0.714	0.833	0.769	0.796	0.817	0.806
Photography	0.742	0.821	0.780	0.813	0.807	0.810
Education	0.727	0.828	0.774	0.781	0.810	0.795
Average	0.728	0.827	0.774	0.800	0.810	0.800

complex sentence patterns lead to incorrect FModifier information; (3) errors in dividing phrases cause some FModifiers to be wrongly divided into multiple FModifiers. Besides the above metrics, we are also concerned about how many of the extracted FModifiers are details of useful App functions. Thus, we further analyse the extracted data and evaluate whether they are useful, and the evaluation result is also shown in Table 7. It can be found that 45.23% of the gained results are related to details of App functions.

Third, Table 8 and Table 9 present the influence of operations on the information extraction. From Table 8 it can be found that the operation extending the FModifier can only slightly improve the accuracy of extracting MFs in descriptions (0.0631), but the operation non-function dictionary filtering can improve the result greatly (0.1765). Compared with precision, the operations cannot change the recall of MF extraction effectively. Additionally, it can be seen from Table 9 that both non-detail dictionary filtering and dividing the FED can improve the precision of FModifier extraction (0.072 and 0.05, respectively), and the recommendation of two operations reduces the recall value. There seems to be a trade-off between precision and recall after adding operations into the process of information extraction, but considering the changes in the F-measures, we believe that operations can have a positive effect.

To further analyse the performance of our method, we apply our dataset to two existing feature extraction methods (SAFE and Colocation) and aim to compare them with our method on a common data. Table 10 shows the experimental results of the compared methods and our method. It can be found that the recall of our method is slightly lower than that of the compared methods, but the precision value is far higher than them. Thus, we believe that our method can complete the task of feature extraction effectively.

4.3.1 | Experiment for sub-question (2)

To assess whether the clustering method can cluster similar features together, we first determine the values of A, B and c in formula 1 and then identify the thresholds for judging the similarity of OPE and OBJ in the features. Finally, we randomly select some features from Apps in the Social category and cluster them by using the DBSCAN algorithm and further analyse the clustering results.

As for identifying the suitable value of A, we first invite an evaluator to randomly select 100 pairs of relevant nouns and 100 pairs of irrelevant nouns from the App descriptions in the Social category. Then, another evaluator checks these

TABLE 10 The experimental results of compared methods

	Main subject of feature extraction			Modifier of feature extraction			SAFE			Colocation		
Category	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**
Social	0.6966	0.7654	0.7294	0.840	0.783	0.810	0.3543	0.8128	0.4935	0.3333	0.8610	0.4806
Photography	0.7243	0.7204	0.7223	0.846	0.786	0.815	0.4437	0.7882	0.5678	0.4083	0.8847	0.5587
Education	0.7117	0.6810	0.6960	0.865	0.776	0.818	0.4153	0.8583	0.5598	0.3408	0.7625	0.4710
Average	0.7109	0.7223	0.7159	0.850	0.782	0.814	0.4044	0.8198	0.5404	0.3608	0.8361	0.5034

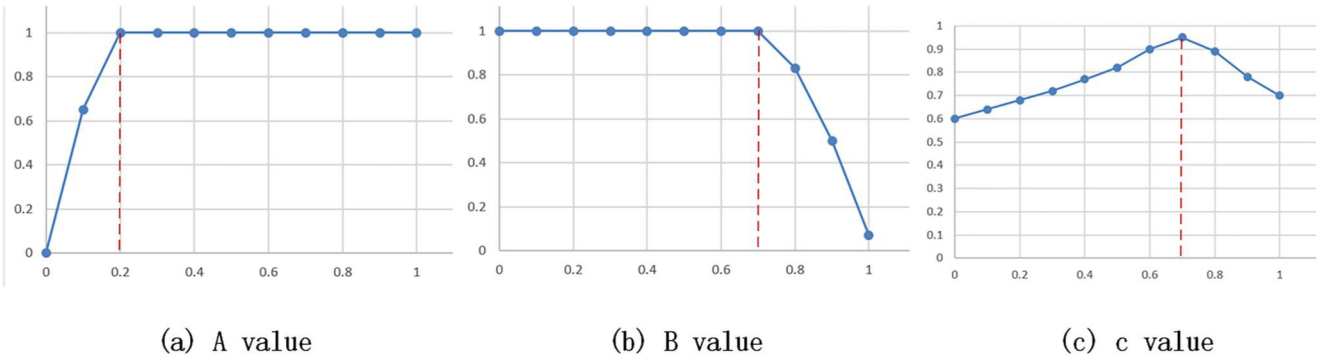

FIGURE 7 The result of determining the values of A, B and c

nouns, and for the disagreements, they need to discuss and reach a consensus. Using 0.1 as the gradient, we test the value in the range from 0 to 1 for determining the optimal value of A. It can be seen from Figure 7 (a) that the pairs of irrelevant nouns can be mined when A = 0.2. In addition, we use the same method to identify the suitable value of B; Figure 7 (b) shows that the pairs of relevant nouns can be found when B = 0.7.

With an aim of identifying the proper value of c, an evaluator randomly selects 100 groups of OPEs and 100 groups of OBJs from the descriptions in the Social category. Each group consists of a pair of similar phrases and three irrelevant phrases. Another evaluator checks these phrases, and for the disagreements, they need to discuss and reach a consensus. Also using 0.1 as the gradient, we test the value in the range [0,1] to identify the value of c, which makes the results gained by our method similar to those given manually. From Figure 7 (c) it can be observed that it has the highest accuracy when c = 0.7.

Only when the OPE and OBJ in the features are similar at the same time can we think that the features are similar. Thus, to determine the similarity threshold of the OPE and OBJ, we first invite an evaluator to randomly select 100 pairs of OPEs and 100 pairs of OBJs from App descriptions in the Social category (half of the samples are relevant and half are irrelevant) and then invite another rater to check these samples. Using 0.1 as the gradient, we identify proper thresholds for the OPE and OBJ in the range of 0 to 1. Figure 8 shows that the proper values of these two thresholds are 0.5 and 0.6, respectively.

After determining the parameters used to calculate the similarity between features, we further evaluate the performance of the clustering method. First, an evaluator randomly selects 30 features from the social Apps and divides them into different categories according to their semantic meanings. Another evaluator checks these results and discusses with the first evaluator regarding disagreement. It can be seen from Figure 9 that 30 features are divided into seven categories. Then, we cluster these features by using our method and compare the results with the result given by evaluators to assess the performance of our method. Figure 9 shows the comparison between the clustering results of our method and the evaluators' clustering results. It indicates that the results of the comparison are relatively similar on the whole, but there are still some differences. The main difference between the results obtained by automated clustering and those obtained by the evaluators is that feature 11 (create images), feature 12 (decorate images), feature 21 (send photos) and feature 22 (send images) are separated from the original clusters and gathered into a new cluster. Two situations can be utilised to explain this difference —one is that the cluster containing feature 11 and 12 is indeed similar to the cluster containing feature 21 and 22, while the other is that the amount of texts used to train word2vec is relatively small so that the similarity calculation between some words cannot obtain accurate results.

4.4 | The survey for question 2

Evaluating the usefulness of our method is a non-trivial task since it may be subjective, and it can better reflect the applicability of our proposed method in the development

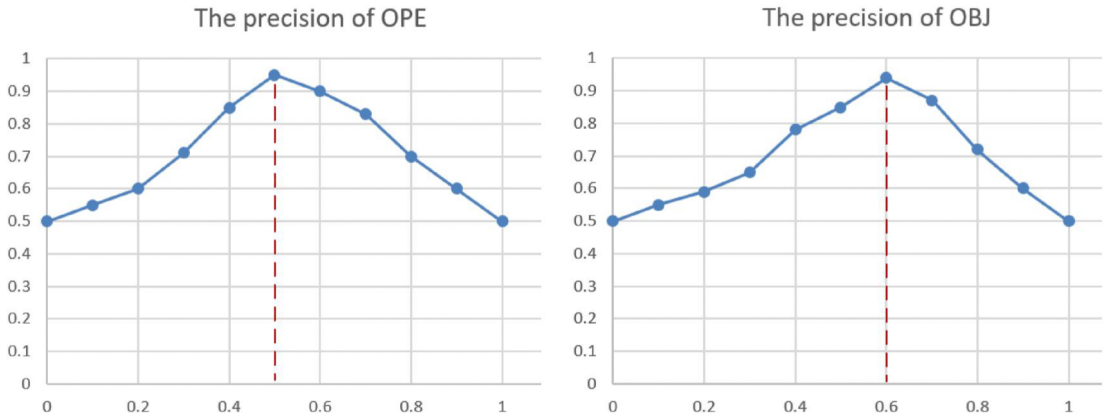

FIGURE 8 The result of determining the thresholds of the OPE and OBJ

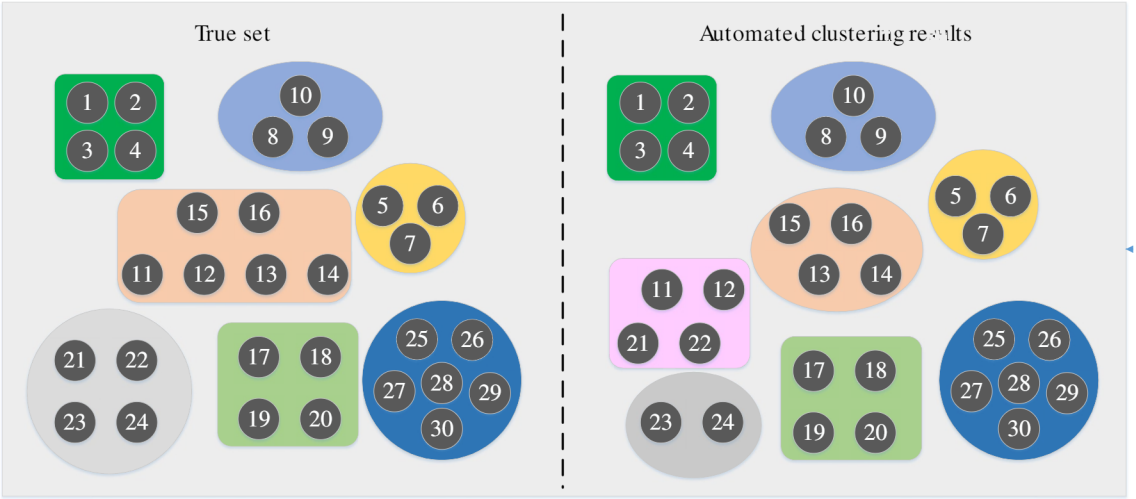

FIGURE 9 The clustering result of features

Question	Question content
Question 1	Whether adding FModifier into the feature can enrich detailed information of App functions?
Question 2	Whether the feature with detailed information can help compare App functions deeply?
Question 3	Whether our method can inspire developers to generate ideas when they design their products?
Other feedback	Any other feedback?

TABLE 11 The design of the questionnaire

process. In this process, we cannot establish a truth set to calculate quantitative evaluation parameters. Thus, we conduct an investigation on 20 developers to answer Question 2.

4.4.1 | Design the survey

First, we extracted features from the App descriptions, see Table 5. Then, we clustered the features of each category and divided the detailed information of clusters. Finally, we designed a questionnaire to obtain feedback from developers. We selected 10 clusters from each category and conveyed them to 20 developers and consulted the developers to evaluate the information provided by us.

The questionnaire is shown in Table 11. Q1, Q2, and Q3 aim at evaluating the information of the feature, the usefulness of the feature, and the inspiration from our method, respectively. Please note that, as the participants do not have much time, our questionnaire in the survey is simple with only quantitative questions. We provide five options in each question (1 strongly disagree, 2 disagree, 3 neither, 4 agree

TABLE 12 The results of the questionnaire

Question	Strongly disagree	Disagree	Neither	Agree	Strongly agree	Total
Q1	0	0	0	14	6	20
Q2	0	0	2	13	5	20
Q3	0	1	5	12	2	20

and 5 strongly agree). Furthermore, a square frame named 'Any other feedback?' is given at the end of the questionnaire so that the participants can give any comments or ideas freely.

4.4.2 | Feedback of the survey

Table 12 summarises the feedback from the survey, and we then analyse it from three perspectives:

Firstly, we find that all the participants agree that adding the FModifier into the feature can enrich the detailed information about App functions. This means that our feature extraction method can get more information than other methods.

Secondly, most of the participants (90%) deem that the feature with detailed information can compare similar App functions thoroughly. Indeed, the dividing result of the FModifier can further show how other App products realise similar functions.

Thirdly, these results also indicate that only 14 of 20 participants (70%) agree that our method can encourage developers to get inspiration when developing Apps, but this ratio is much lower than the results of the above two questions. After analysing the contents in square frames, we find that it is mainly because that many participants did not get the detailed information they wanted, and they suggest us to extract App features from more App-related text so as to give them more reference when designing or upgrading Apps.

To further illustrate the opinions of participants, we show some feedback in the square frames as follows, including encouraging comments and valuable suggestions:

Encouraging comments:

> 'I believe this work can help developers when they design App functions.'

> 'Extracting and comparing App features can save my time comparing similar Apps.'

> 'It gives me a new way to learn valuable new features. For example, finding lost mobile phones is a common function, but through your method, we get the details of finding lost mobile phones of others, which obviously can make App stand out in similar products.'

Valuable suggestions:

> 'A good job, I care not only what features my competitors have but also what features users need. We hope XXX can show some information about user feedback.'

> 'I cannot understand some dividing result of FModifier, please filter the noise data to have a better readability result.'

> 'I hope your method can show what new functions appear in a period of time, and through our method, we can understand the development of the domain in a period of time, so as to provide better help for the iterative process of App updating.'

In summary, the participants in our experiment confirm the usefulness of our approach in practice. The detailed information obtained by our method can not only help developers compare the App functions but also help them make updated strategies of products and encourage them to get inspiration.

4.5 | Threats to the validity of experiments

Despite the encouraging results, this work has some potential threats to validity. We provide a discussion on them from the following three aspects:

The dataset used in our approach is limited. The dataset in our experiments is established based on Google Play, and so it is unclear whether our work can achieve similar results in other App stores. Due to the complexity of our experiments, we could not evaluate our approach with additional datasets in a short time. However, the form of data in App stores is similar; so we believe that our approach can also be suitable for them.

The method used to compare the App cannot be evaluated directly. We cannot evaluate our method used to compare the App directly because it is difficult to judge whether the App competitive information given by us is correct or not. Thus, we can only design a survey based on our method to get the feedback of developers. Although we interviewed only a few developers, most developers provided positive feedback; hence we believe that our method is helpful in comparing similar Apps.

The reliability of labels of samples in truth sets. The labelling process of samples in truth sets used in Question 1 is complex and needs professional background knowledge. Although three evaluators who labelled the data have certain

background knowledge in the field of App products, there is still a possibility for them to give wrong labels. However, the labels in our truth sets are tagged by multiple evaluators independently, and their results are discussed to converge when discrepancies occurred. Thus, we believe that the labels of samples in our truth sets are reliable.

5 | RELATED WORK

In this study, since the comparison of App functions can help developers to make competitors analysis and mine domain knowledge, in this section, we will discuss the related studies from the two following aspects:

5.1 | Competitor analysis

Competitor analysis focusses on comparing a product with existing products in the market before development, and the existing methods can find the competitor among the organisation, product brands and persons. In the early research studies, the related works are more focussed on identifying the competition, such as identifying the comparative sentences [13], and objects of comparison are often the competitors. Shenghua Bao et al. [14] found competitors according to the competitive evidence which is the comparative sentence on the web, then they introduced the competitors' strength in detail. George Valkanas et al. [15] judged the competitiveness between two items according to whether they competed for the same groups of customers, and then they also compared competitors' features that users are interested in. Although these methods can determine the competitive relationship in the market, and the technologies they used were still suitable for the App competitor determined, the developers were more concerned about the details of App functions. In recent years, there are many available data to support the competitor analysis; the researchers began to focus on dealing with these data and compare the competitor under different topics. To mine the text content on Facebook and Twitter, Wu He et al. [16] established some themes to classify the textual data from customers; the text in each theme can help marketers get users' feedback and understand what their competitors are doing. Qiang Wei et al. [17] used query logs from search engines to measure the competitiveness degree between peers, and they compared similar products according to the aspects that users pay attention to. Yang et al. [18] mined competitive relationships by learning across Twitter and patent records; they analysed the companies' competition in different aspects and compared them at the topic level. Ning Chen et al. [19] and Srisopha et al. [9] used the LDA, a classical topic modelling method, to classify App reviews into different topics. Comparing Apps under different topics can provide developers with some information, but limited topics and a few words under these topic cannot provide developers with specific opinions.

There are some open platforms to bring all Apps together and classify them into different categories; so some studies of Apps in such competitive markets have emerged. H. Jiang et al. [20] extracted App features and API to find similar Apps and helped developers identify new features. Faiz Ali Shah et al. [21] identified App competitors according to the features in App reviews and compared Apps at the feature level [2, 22]. These studies regard the Apps in App stores as competitors, they discover common features among Apps and give developers suggestions at the feature level. However, they cannot provide more specific information about the features for developers. There are some studies focussed on the UI of Apps, and they also collected data from similar Apps. Chen et al. [23] proposed a system to generate the storyboard for Android Apps automatically in Google Play and F-Droid; it could provide reference for the developer, the UI/UX designer and developers to develop the product. Xiangping Chen et al. [24] mined features from similar UIs provided by other Apps, and offered some valuable functional requirements to developers. Similar UI provided by these studies may contain detailed information, but the detailed information is not separated from the common feature. When the developer needs to study the App features in depth, they are still required to manually find the detailed information about the App functions.

5.2 | Feature extraction

App descriptions, as the important information that briefly introduces the main functions of this App, is the main way to help users have a preliminary understanding of it. At the same time, it is also an important way to explore the App, such as extracting features from it, generating the descriptions, and so on. In this work, we concentrate on how to extract features from descriptions, and there are some existing studies that can solve this problem. They use different feature extraction methods for different purposes. To support App Store mining and analysis, Harman et al. [6] identified features in the App descriptions according to some pattern templates, and performed a word frequency and co-location analysis to extract such information, then they further studied the life-cycles of features [25] and predicted App rating using App features [26]. Huayao Wu et al. [27] applied a textual pattern-based approach and classifier to extract features from Chinese App descriptions, determined user reviews that are relevant to features, and applied a multi-linear regression model to identify key features. To find the relationship between App description and App review, Timo et al. [7] defined 18 part-of-speech patterns and extracted features from App descriptions and reviews according to those patterns. Different from the above studies, we not only focus on the features of the App, but also pay attention to the details of these features to complete the task of App function comparison in the competitor analysis, and these detailed information are often ignored by most of the existing methods. Although to a certain extent, the features extracted by Timo et al. [7] contain some of the details of App functions, they did not separate the body of App functions

from the details of App functions; so the features also cannot be further compared. To the best of our knowledge, there are few feature extraction methods that can extract the details of App functions.

App reviews are another important text resource in App stores, through which users can express their feedback, requirements, and appreciation of the App [28]. Emitza et al. [29] focussed on the features in specific App descriptions that can be visible for users and then used co-location to automatically extract App features referred to in the reviews together with the users' opinions to achieve their purpose. Lorenzo et al. [5] extracted features in App reviews using n-gram. They classified user reviews based on extracted features and feedback to developers. Fabio Palomba et al. [8] extracted verb–noun pairs as a feature after applying a part of speech tagging the reviews. They used such features to locate users' feedback in the code. Haroon Malik el al. [30] extracted features and opinions from App reviews according to the dependency rules of the sentence, and then they compared Apps by identifying 'Hot' features.

These works can extract features of App functions from App reviews. But in our work, we find that App reviews are not standardised; so the accuracy of extracting features from App reviews is lower than extracting features from App descriptions. At the same time, the features in the reviews are too subjective and some features are simple and one-sided; so they cannot describe the App functions clearly and it is difficult to make a comparison with them. Thus, different from the above works, the features we extracted not only contain the functions of the App but also combine the detailed information of these functions and the corresponding competitive information, which can better help developers make the comparison.

6 | CONCLUSION AND FUTURE WORK

To help developers analyse the Apps that have similar functions and encourage them to design a more unique plan when developing and updating Apps, this work proposes a method to automatically extract features with detailed information from descriptions and integrate the results to assist developers in the process of development. First, we divide the descriptions from different perspectives (the subject and modification of a feature), and then we determine the feature components and combine them. Thus, we propose a method to extract features with more detailed information. Second, we also propose an approach to evaluate the similarity between features and then cluster them of one aspect together. Third, we divide the detailed information in the clustering result for each feature by matching the keywords.

To evaluate our method, we carry out large-scale experiments and deeply explore the corresponding results. The results of the experiment show that our method can mine the feature information and integrate them effectively (F-measure = 71.6%). Further, the feedback from 20 developers indicates that our method shows great usefulness and operability both in the early stages of application development and later version updates (Agree: 13/20). At the same time, developers can use this method to compare with other Apps and formulate competitive analysis schemes.

Our future studies will mainly focus on two aspects. First, we want to develop a well-established tool to support our approach in practice. Second, we hope to evaluate our approach in more cases and help developers to analyse their Apps.

ACKNOWLEDGMENTS

This work is funded by the National Key Research and Development Programme of China 2017YFB1003103, the Natural Science Research Foundation of the Jilin Province of China under Grant Nos. 20190201193JC, the Graduate Innovation Fund of Jilin University 101832020CX181, and the Interdisciplinary Research Funding Programme for Doctoral Students of Jilin University 101832020DJX064.

DATA AVAILABILITY STATEMENT

https://zenodo.org/record/4346585

ORCID

Huaxiao Liu https://orcid.org/0000-0002-8151-1413
Shanshan Song https://orcid.org/0000-0002-5762-8160

REFERENCES

1. Number of Available Applications, https://www.statista.com/statistics/266210/num-ber_of_available-applications-in-the-google-play-store/_September_2020
2. Shah, F.A., Sirts, K., Pfahl, D.: Using app reviews for competitive analysis: tool support. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics (WAMA) Association for Computing Machinery, New York (2019)
3. Stanik, C., et al.: Which app features are being used? Learning app feature usages from interaction data. In: IEEE 28th International Requirements Engineering Conference (RE), pp. 66–77. (2020)
4. Competitor analysis before launching a mobile app startup, https://growthbug.com/competitor-analysis-before-launching-a-mobile-app-startup-f2f6a19f21b7
5. Villarroel, L., et al.: Release planning of mobile apps based on user reviews. In: IEEE/ACM 38th International Conference on Software Engineering (ICSE), 14–24 (2016)
6. Harman, M., Jia, Y., Zhang, Y.: App store mining and analysis: MSR for app stores. In: 9th IEEE Working Conference on Mining Software Repositories (MSR), pp. 108–111. (2012)
7. Johann, T., et al.: SAFE: a simple approach for feature extraction from app descriptions and app reviews. In: IEEE 25th International Requirements Engineering Conference (RE), 21–30 (2017)
8. Palomba, F., et al.: Recommending and localizing change requests for mobile apps based on user reviews. In: IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp. 106–117. (2017)
9. Srisopha, K., et al.: On building an automatic identification of Country-specific feature requests in mobile app reviews: possibilities and Challenges. In: IEEE/ACM 42nd International Conference on Software Engineering (ICSE) (2020)
10. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. SGMD (1999)
11. Ester, M., et al.: A density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. KDD (1996)
12. Bagga, A., Baldwin, B.: Entity-based cross-document coreferencing using the vector space model. In: Meeting of the Association

for Computational Linguistics and, International Conference on Computational Linguistics. Association for Computational Linguistics (1998)
13. Jindai, N., Liu, B.: Identifying comparative sentences in text documents. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR) (2006)
14. Bao, S., et al.: Competitor Mining with the Web. IEEE Trans. Knowl. Data Eng. 20, 1297–1310 (2008)
15. Valkanas, G., Lappas, T., Gunopulos, D.: Mining competitors from large unstructured datasets. IEEE Trans. Knowl. Data Eng. 29, 1971–1984 (2017)
16. He, W., Zha, S., Li, L.: Social Media competitive analysis and text mining: a case study in the pizza industry. Int. J. Inf. Manag. 33, 464–472 (2013)
17. Qiang, W., et al.: A novel bipartite graph based competitiveness degree analysis from query logs. ACM Trans, Knowl. Discov. Data (TKDD), 11, 1–25 (2016)
18. Yang, Y., Tang, J., Li, J.-Zi.: Learning to infer competitive relationships in heterogeneous networks. ACM Trans. Knowl. Discov. Data (TKDD). 12, 1–23 (2018)
19. Chen, N., et al.: AR-miner: mining informative reviews for developers from mobile app marketplace. In: IEEE/ACM 36th International Conference on Software Engineering (ICSE). pp. 767–778. (2014)
20. Jiang, H., et al.: Recommending new features from mobile app descriptions. ACM Trans. Softw. Eng. Methodol. (TOSEM), 28, 1–29 (2019)
21. Sjaj, F., Sabanin, Y., Pfahi, D.: Feature-based evaluation of competing apps. In: Proceedings of the 1st ACM SIGSOFT International Workshop on App Market Analytics (WAMA), pp. 15–21. (2016)
22. Gu, X., Kim, S.: What parts of your apps are loved by users? (T). In: 2015 30th IEEE/ACMInternational Conference on Automated Software Engineering (ASE). pp. 760–770. (2015)
23. Chen, S., et al.: StoryDroid: automated generation of storyboard for android apps. In: IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 596–607. (2019)
24. Chen, X., et al.: Recommending software features for mobile applications based on user interface comparison. Requir. Eng. 24, 545–559 (2019)
25. Sarro, F., et al.: Feature lifecycles as they spread, Migrate, remain, and die in app stores. In: IEEE 23rd International Requirements Engineering Conference (RE), 76–85 (2015)
26. Sarro, F., et al.: Customer rating reactions can Be predicted purely using app features. In: IEEE 26th International Requirements Engineering Conference (RE), pp. 76–87. (2018)
27. Wu, H., et al.: Identifying key features from app user reviews. In:2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 922–932. (2021)
28. Stanik, C., Häring, M., Maalej, W.: Classifying multilingual user feedback using traditional Machine learning and deep learning. In: IEEE 27th International Requirements Engineering Conference Workshops (REW), pp. 220–226. (2019)
29. Guzman, E., Maalej, W.: How do users like this feature? A fine grained sentiment analysis of App reviews. IEEE 22nd International Requirements Engineering Conference (RE), 153–162 (2014)
30. Malik, H., Shakshuki, M., Yoo, W.-S.: Comparing mobile apps by identifying Hot features. Future Gen, Computer Syst. (FGCS), 107, 659–669 (2020)

How to cite this article: Liu, H., et al.: Mining detailed information from the description for App functions comparison. IET Soft. 16(1), 94–110 (2022). https://doi.org/10.1049/sfw2.12042