The Journal of Systems and Software 138 (2018) 108–123

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Automatically classifying user requests in crowdsourcing requirements engineering

Chuanyi Li[a], Liguo Huang[b], Jidong Ge[a,*], Bin Luo[a], Vincent Ng[c]

[a] State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, 210093, China
[b] Department of Computer Science and Engineering, Southern Methodist University, Dallas 75275-0122, TX, USA
[c] Human Language Technology Research Institute, University of Texas at Dallas, Dallas 75083-0688, TX, USA

ARTICLE INFO

Article history:
Received 4 January 2017
Revised 17 December 2017
Accepted 19 December 2017
Available online 20 December 2017

Keywords:
Crowdsourcing requirements engineering
Software requirements classification
Machine learning
Natural language processing

ABSTRACT

In order to make a software project succeed, it is necessary to determine the requirements for systems and to document them in a suitable manner. Many ways for requirements elicitation have been discussed. One way is to gather requirements with crowdsourcing methods, which has been discussed for years and is called crowdsourcing requirements engineering. User requests forums in open source communities, where users can propose their expected features of a software product, are common examples of platforms for gathering requirements from the crowd. Requirements collected from these platforms are often informal text descriptions and we name them user requests. In order to transform user requests into structured software requirements, it is better to know the class of requirements that each request belongs to so that each request can be rewritten according to corresponding requirement templates. In this paper, we propose an effective classification methodology by employing both project-specific and non-project-specific keywords and machine learning algorithms. The proposed strategy does well in achieving high classification accuracy by using keywords as features, reducing considerable manual efforts in building machine learning based classifiers, and having stable performance in finding minority classes no matter how few instances they have.

© 2018 Elsevier Inc. All rights reserved.

1. Introduction

Software Requirements Engineering (**RE**) is a systematic and discipline approach to the specification and management of requirements. It is a very important phase in the software development process and regardless of whether it belongs to a ponderous process model (e.g., the Waterfall model (Royce, 1970) or the V-Model (V-Modell, 2004) or a lightweight process model (e.g., eXtreme Programming (Beck, 2000)). According to the Standish Group's Chaos Report of 2015 (Chaos, 2015), among all 8380 applications investigated, the success rate was only 16.2%, while challenged (i.e., the project is completed and operational but over-budget, over the time estimate, and offers fewer features and functions than originally specified) projects accounted for 52.7%, and impaired (cancelled) for 31.1%. The three major reasons that a project will succeed are user involvement (15.9%), executive management support (13.9%), and a clear statement of requirements (13%). The three major factors that cause projects to be challenged are lack of user input (12.8%), incomplete requirements and specifications (12.3%) and changing requirements and specifications (11.8%). Opinions about why projects are impaired and ultimately cancelled ranked incomplete requirements (13.1%) and lack of user involvement (12.4%) at the top of the list. Almost all of these factors are related to software requirements engineering directly. This is solid evidence that the way the requirements of a system are handled is a significant cause for project failures (Pohl and Rupp, 2011). According to past studies, approximately 60 percent of all errors in system development projects originate during the phase of requirements engineering (Boehm, 1981). The ultimate goals of requirements engineering are to achieve a consensus among the stakeholders of their desires and needs on the product, and to document these requirements according to given standards. One of the most important artifacts in requirements engineering is specification, which plays a key role through the entire software development process. In order to write the requirements specification, desires and needs of stakeholders towards the product should be collected through meetings or any other communicating way firstly. This fundamental phase is called requirements elicitation.

In Snijders et al. (2015), it is concluded that requirements elicitation is one of the most concerned work in requirements engineering, and the benefits of involving customers and users of products in elicitation have been widely acknowledged since

* Corresponding author.
E-mail address: gjd@nju.edu.cn (J. Ge).

https://doi.org/10.1016/j.jss.2017.12.028
0164-1212/© 2018 Elsevier Inc. All rights reserved.

Zand and Sorensen (1975) showed that user participation helps in overcoming resistance to change. However, customers and users are often geographically distributed (Cheng & Atlee, 2007) and technical and social environments are uncertain (Hosseini et al., 2014). Besides, as the context of software usage changes, it is hard to determine what kind of users will use the system in which environment and on which device. Applying crowdsourcing which is the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined network of people in the form of an open call (Howe 2006) might help in addressing these issues. So, crowdsourcing requirements engineering (i.e., gathering software requirements from different stakeholders, customers or users of the product through the Internet) is becoming an important way for software requirements elicitation. Since every single stakeholder or customer gets an opportunity in proposing his/her expectations of the products, the gathered requirements will be more completed than those generated by only held meetings among majority stakeholders. The data gathered by crowdsourcing requirements engineering will be used as input for discussing and documenting software specifications. In this paper, we refer to the data submitted by stakeholders or customers as user requests.

Since normal users are not trained or skilled experts, requests submitted by them are often informal natural language descriptions. Before adding user requests into the software requirements specification, they should be rewritten according to the given rules or templates of software requirements. However, in software requirements specifications, requirement items are grouped according to their types, e.g., capability, usability, security and performance, etc. Different types of requirements have different rules or templates. Thus, to rewrite user requests efficiently, we should firstly find the correct type of requirements it belongs to. This is called user requests classification. Except for having the benefit of transforming user requests into structured requirements, classification also helps to handle duplications among user requests precisely. First, requests have a same topic but different detailed concerns would be classified into different classes and this avoids regarding them as duplications. Second, finding duplications with each single class of requests makes it more efficient. In open source projects, where rewriting user requests into structured requirements is not necessary, classifying them into different classes will help project members determine handling sequence of requests according to priorities of different requirements types. For example, security or reliability requests are often the more important than capability or usability ones. However, manual classification is time-consuming, and for large projects, error prone. So, automated classification strategies, even partial, will be of great benefit (Vlas and Robinson, 2012).

In crowdsourcing requirements projects, there may be more than thousands of stakeholders and consumers and the number of gathered requests can be huge. Vlas and Robinson (2012) propose a rule-based natural language strategy for requirements discovery and classification in open source software development projects. A 2006 study by Cleland-Huang et al. (2006, 2007) also explores approaches to requirements discovery and classification. Their work (i.e., processing the unstructured natural language user requests with NLP techniques) is useful for detecting potential software requirements from all user submitted user requests. In our research, we focus on the problem of classifying detected potential requirements (i.e., user requests) into corresponding requirements types. Different from Vlas and Robinson (2012) and Cleland-Huang et al. (2006, 2007), we make use of both project-specific and non-project-specific keywords, and employ machine-learning techniques in our approach. In summary, the main contribution we made in this paper is proposing a keywords-based machine approach to semi-automatically classify user requests in crowdsourcing scenarios. The approach consists of four major parts, namely existing machine learning algorithm SVM, non-project-specific and project-specific keywords, heuristic properties of user requests, and active learning strategy.

For evaluating the proposed approach, we present results of experiments conducted on data sets collected from open source user request forums. These experiments are designed to answer the following six research questions:

RQ1. How well can those user requests be classified by only using widely used simple text features as input of machine learning algorithms, e.g., word unigram and TF-IDF?

RQ2. Will the classification accuracy of entire data set as well as minority classes be improved by using keywords-based features?

RQ3. To what extent are the proposed non-project-specific keywords accepted by others? Are they more effective than those proposed by others?

RQ4. Which machine learning algorithm that widely used in requirements classification does best here, k-nearest neighbor, Naïve Bayes or Support Vector Machine?

RQ5. Do the proposed Heuristic Properties of user requests help improve the classification accuracy?

RQ6. Can keywords-based features help reduce the size of training sets to achieve the same accuracy derived by using widely used features?

The remainder of this paper is laid out as follows. Section 2 introduces related work. Section 3 describes the classes of user requests defined by us and two kinds of keywords found by us. Section 4 introduces our approach in detail and Section 5 presents experiments results and answers research questions, including a subsection for discussing the entire study from different aspects. Section 6 analyzes threats to validity of our approach. Section 7 concludes with a discussion of future work.

2. Related works

We next present a summary of existing studies on crowdsourcing requirements engineering, software requirements classification, and techniques used in natural language document classification.

2.1. Crowdsourcing requirements engineering

Crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call (Howe 2006). In Brabham (2008), Crowdsourcing is defined as an emerging, typically online, distributed problem solving and production model where a problem is solved through the involvement of a large number of people. Many different crowdsourcing systems on the Word-Wide Web were discussed in Doan et al. (2011), who show that crowdsourcing could be applied to a wide variety of problems. It was concluded that more generic platforms, more applications and structure, and more users and complex contributions were the three major developing directions of crowdsourcing systems. As the uncertainty of technical and social environments has increased radically due to the ever-increasing utilization of web-based systems, system developers and software engineers encounter a wider audience of users, which we call the general public, or the crowd. To cater the requirements of the crowd, it would be helpful to apply crowdsourcing in requirements elicitation. Crowdsourcing in requirements elicitation has the potential to increase the quality and comprehensiveness and even the economic feasibility of requirements elicitation. The potential was investigated in Hosseini et al. (2014),

who explore the relationship between the features of main constructs and quality of requirements elicited via crowdsourcing. A more detailed Crowd-Centric Requirements Engineering Method is proposed in Snijders et al. (2015), who answer the question of how crowdsourcing can be used to improve requirements engineering in software production, including in which activities and to what extent the quality of RE can be improved. Observing that the participants with a particular kind of domain knowledge often have the opportunity to cluster in particular spatiotemporal spaces, a novel opportunistic participant recruitment framework was proposed in Wang et al. (2014) to enable organizers in crowdsourcing RE to recruit participants with desired kind of domain knowledge in a more efficient way. Specialized crowdsourcing systems for requirements elicitation are also discussed. A generic RE crowdsourcing platform, CrowdREquire, is introduced in Adepetu et al. (2012), where the business model, purpose, goals, stakeholders, and context are well defined. Finkelstein and Lim (2012) introduced a way to use social networks together with collaborative filtering technique to elicit large-scale requirements from the crowd. In order to support rich data collection and volunteer participation effectively and efficiently, high-level conceptual functionality and usability requirements for a crowdsourcing task interface are produced in McKinley (2013). Many commercial software firms rely on open source software (OSS) communities as a source of innovation and skilled labor (Naparat and Finnegan, 2013). One specific form of interaction with open source software (OSS) communities, termed 'opensourcing', involves firms collaborating with an OSS community by 'crowdsourcing' software production. In Naparat and Finnegan (2013), how firms can collaborate with communities to crowdsource the production of software was explained. Different from others, Breaux and Schaub (2014) designed and conducted concrete experiments to evaluate quality of crowdsourcing by scaling requirements extraction tasks to the crowd.

2.2. Natural language processing in software requirements engineering

One of the most important outputs of software requirements engineering is requirements specifications that are written in well-defined and structured templates. Structured requirements templates do well in avoiding ambiguities that may arise from the use of unrestricted natural language (Arora et al., 2015). However, most artifacts derived from requirements elicitation stage are described in unstructured natural language. Transforming requirements from free-form natural language into structured templates manually is laborious. So, many natural language processing (NLP) techniques are introduced in processing exiting requirements-related artifacts or documents, such as user guidelines, user manuals, and request for proposals, to help in classifying sentences or retrieving information from sentences for generating structured software requirements (Haiduc et al., 2016). Natural language document classification techniques, which typically involve machine learning algorithms such as k-nearest neighbor (k-NN) (Keller et al., 1985), Naïve Bayes (NB) (Rish, 2001), and Support Vector Machine (SVM) (Suykens and Vandewalle, 1999), are widely used in requirements artifacts classification (Slankas and Williams, 2013; Riaz et al., 2014). To represent requirements documents in classification algorithms, a bag of words (BOW), such as word unigram, bigram, n-grams, and term frequency-inverse document frequency (TF-IDF), is often used. Other NLP techniques, such as stopwords removal, stemming, lemmatization, tense detection, are used in pre-processing these documents (Maalej and Nabil, 2015). In Slankas and Williams (2013), the Stanford Natural Language Parser is used to pre-process each sentence in existing documents and outputs a graph in the Stanford Type Dependency Representation (STDR) based on the Part-of-Speech (POS) tag of each word. In Galvis and Winbladh (2013), a Topic Modeling algorithm, Latent Dirichlet Allocation (LDA), is used to derive evolved user requirements from user comments that are described in unrestricted natural language. In this paper, we also apply pre-processing methods on input documents, use different representations of them, and try to classify them with different machine learning algorithms.

2.3. Software requirements classification

One of the most important tasks in software requirements engineering is requirements classification. In research on the software requirements lifecycle, a lot of works have been done on defining reasonable and comprehensive classes of software requirements. Well-categorized requirement items would improve the quality of requirement specification, which in turn improves the quality of software (Pohl and Rupp, 2011). In order to reduce the difficulty in writing well-categorized requirements documents, techniques for classifying elicited requirements into different classes automatically are needed. Since we know how to determine the class of a requirement, we could also retrieve it from a long document according to its characters. Based on this idea, many methods for classifying and retrieving a specific class of requirements from existing software artifacts are developed, such as security requirements (Slankas and Williams, 2013; Riaz et al., 2014) and usability requirements (Ormeño et al., 2013). In the requirements evolving process where requirements are elicited from users in the forms of bug reports, feature requests or simple comments, it is more useful to determine the exact class of each requirement, because different classes of requirements would have different priorities in implementation in the next product release. So, in Casamayor et al. (2010), Vlas and Robinson (2012) and Zhang et al. (2011), different classification methods for user submitted requests are proposed.

3. Classes of requirements

As illustrated in Introduction section, classifying user requests into different types benefits in three ways, namely, writing well-structured requirements specification, handling duplicated requirements, and advising developers about sequence of implementing features. However, we find there is no consensus on detailed destination classes for classifying software requirements in existing research. As shown in Table 1, there are many different classifications of software requirements. Considering the ultimate goals of user requests classification and normal structures of requirements specification, there is no need to classify requests into all mentioned classes of requirements. So, we group these classes into seven types (i.e., first column in Table 1) according to their priorities considered by developers while implementing requests proposed by users. Although the types defined by us are groups of existing requirements classes, we try our best to provide general definitions to make it easier for readers to understand. The definitions of all types of user requests are:

- **Security** (SE): Security refers to requirements discussing attacks on purpose that threaten either the system or information safety in all kinds of ways.
- **Reliability** (RE): Reliability refers to the robustness, tolerance, predictability and recoverability of the system. More generally, it is the description of the frequency or severity of a system failure, the accuracy of system reaction and its ability to recover from failures.
- **Performance** (PE): Performance is related to concerns on system speed, efficiency, resource consumption and throughput, etc.
- **Lifecycle** (LI): By lifecycle, we mean the requirements of the development of the project instead of the software itself, such

Table 1
Scopes of each user request type referring to exiting software requirements classes.

Type	SERC 2017	Slankas and Williams 2013	Pohl and Rupp 2011	Rashwan et al. 2013	Glinz 2007	ISO/IEC Standard 9126–1 2001	Mylopoulos et al., 1992	Grady and Caswell 1987	Roman 1985	Boehm 1978
Security	Safety Security	Privacy Security	–	Security	Security	Security	–	Security	Security	–
Reliability	Reliability	Recoverability	–	Reliability	Reliability	Reliability	Reliability	Reliability	Reliability	Reliability
	Robustness	Reliability				Accuracy	Survivability		Survivability	
	Survivability						Correctness			
	Repairability									
	Endurability etc.									
Performance	Performance	Performance	Performance	Efficiency	Performance	Efficiency	Efficiency	Performance	Time/space bounds	Efficiency
	Suitability Speed									
	Affordability etc.									
Lifecycle	Maintainability	Maintenance	Portability	Constraints	Portability	Maintainability	Integrity	Supportability	Lifecycle	Portability
	Sustainability	Scalability	Solution space	Maintainability		Portability	Maintainability			Testability
	Changeability		Scalability				Expendability			Modifiability
	Modifiability						Portability			Understandability
	Scalability						Reusability			
	Flexibility etc.									
Usability	Usability Mission-	Look and feel	Availability	Usability	Usability	Usability	Usability	Usability	Operating	Human
	Effectiveness	Operational			Availability					engineering
	Availability	Usability								
	Dependability etc.	Availability								
Capability	Capability	–	Functionality	Functionality	Functionality	Suitability	Functional	Capability	–	–
System Interface	Interoperability	–	–	–	Interface	Interoperability	Interoperability	–	Interface	–

as maintainability, enhanceability, portability, testability, extensibility, adaptability, configurability and installability. Also, it includes constraints on the development process, such as development time limitations, resource availability and methodology standards, etc.

- **Usability** (US): The fundamental discipline of usability is that it starts from human usage considerations. For example, if something should be implemented because it would make it convenient for users to use the software, it is grouped into usability category.
- **Capability** (CA): Generally, capability corresponds to the functional requirements in software requirements engineering. In this paper, it refers to general and basic functions or features of software and is closely related to realistic problems that software should solve.
- **System Interface** (SI): System interface refers to software interfaces, hardware interfaces and definitions of communication methods among people, software and hardware.

Table 2 provides seven simple examples of labeled user requests extracted from the user requests forum from an open source project, *KeePass*,[1] on *sourceforge.net*.[2] Analysts primarily rely on the "Summary" and "Description" sections for user requests classification. A row in Table 2 is a user request proposed by a user to be classified. The "ID" field is only used for uniquely identifying a request but does not contribute to request understanding, and the "Summary" field is optional. If the "summary" is absent, the classification is entirely based on the "description". In order to treat "Summary" and "Description" fairly, we will append the summary to the description as a single sentence before data preprocessing. In order to illustrate the relationship between a request and its label, we also present reasons for labeling a request in Table 2. Since keyword plays a very important role in our approach for classifying user requests, we introduce both non-project-specific and project-specific keywords of each request type next. Non-project-specific keywords are manually derived from analyzing definitions and properties of each type of software requirements, as well as requirements ontologies and taxonomies (Happel and Seedorf, 2006; Dzung and Ohnishi, 2009; Rashwan et al., 2013). Besides, we refer to Wikipedia pages related to introductions of each requirement class and some examples of software requirements specifications. These keywords are collected by four co-authors of this paper following these steps:

(1) everyone was given the same set of above mentioned documents,
(2) each person found keywords of different types of requests according to their own understanding of types' definition and the requirements ontologies and taxonomies found from given documents,
(3) words retrieved by different person were merged according to request type,
(4) we discussed about each word to determine if it can be added into the keyword list ultimately.

It is important to note that if the request contains keywords of one type of requirements, it does not mean that it must be a potential requirement of that type. If the request contains more keywords of one type, it means there is a higher chance that this request should be classified as belonging to that type of requirements than others. Please note that the keywords we collected here may exist in more than one class of requirements. Here are

[1] Homepage of open-source project KeePass is: https://sourceforge.net/projects/keepass/?source=directory.

[2] It is an open-source community and homepage is: https://sourceforge.net/.

Table 2
Examples of features and their classes.

ID	Summary	Description	Class	Reason
1	Just generate password	I have been looking for an app that does not store passwords and just generates them using a hash of a master password phrase and a sort of the hostname of the site.	CA	Request features on key functions of KeePass
2	Linux version	I think a Linux port is a good idea. I use both Windows and Linux but cannot use KeePass on my Linux OS. Please port KeePass to Linux.	LI	Portability: related to the development
3	Only one running instance	Users should only have one instance of KeePass running anytime, or it will cost more system resources. If a user started the program while the program was already running, the program should bring up the already running instance instead of starting a new one.	PE	The number of running instances influence the system performance
4	Color setting is not saved	When I assign a custom background in the Properties Tab of Edit Entry, this custom color is used correctly right now, but it is not kept on the edit entry window is closed.	RE	The setting not saved means a failure
5		At the moment, anyone can change the master key of the database, even if he does not know the original one. It's a big threaten to password safety.	SE	Request on protecting information security
6	Firefox plugin	Please write a Firefox plugin for import passwords from Firefox to KeePass.	SI	Interaction with another software
7		It would be convenient for users to show numbers of entries near the group.	US	Considering convenience of user

examples of non-project-specific keywords of each request type together with illustrations:

- **Security**: Naturally, security concerns are expressed through words related closely to the word "security" semantically. So, words like safe, unsafe, secure, unsecure, safeguard, protect, prevent, against, defend, forbid, warn, attack, harm, offense, offender, risk, threaten, restrict and sensitive, etc., are included. Words for expressing protection methods, such as encrypt, decrypt, read-only, password and confirm, etc., are also added into this list.
- **Reliability**: Requests of reliability often describe the expected consequence that did not happen or unexpected results that happened. In addition to words like reliable, reliability, robust, correct, crash, exception, recovery, fail, wrong, error, problem, incorrect, validate and bug that are derived from its definition and taxonomies, we also add words that are useful in describing an unexpected condition, such as should, but, however, unfortunately, supposed, not, cannot, instead and result in, etc.
- **Performance**: Keywords for performance are mainly derived from its corresponding taxonomies (Van Lamsweerde, 2009), such as space, time, memory, storage, response, throughput, speed, bandwidth, instance, resource, size, accuracy and efficiency, etc. We also add words most related to these nouns, such as limit, save, cost, waste, occupy, improve, reduce, delay, long, short, fast, quick, slow, big, small, large and slow, etc.
- **Lifecycle**: We include the words and their variants in the definition that are related to software process, such as platform, OS, development, testing, debug, maintain, evolve, document, localize, manual, portable, configurable, installation, compile and build, etc. If the user request mentions a name of a system platform, a developing language, a file extension, or an operating system, it is more likely to be a lifecycle requirement. So, the common name of each of these entities are also used as keywords, such as Java, Python, Linux, Windows, Ubuntu, Android, .exe, .dll, etc.
- **Usability**: According to the taxonomies of usability, words like usage, use, usability, message, manually, readability, understandability, documentation, language and operability are included. Besides, words for expressing the feelings of users before or after adding a new feature are also used as keywords, such as useful, handy, convenient, hard, difficult, helpful, annoying, confuse and user-friendly, etc. Since usability requests are often related to GUI and user operations on them, names of UI components and verbs of actions on them are also included. For example, the GUI components contain button, menu, dialog, window, panel and so on. Users' actions contain open, edit, choose, select, press, click, move, close, etc. Keyboard and mouse are also important user interfaces, so names of keys, such as Ctrl, Alt, Shift, and words related to keyboard and mouse, such as shortcut, hotkey, are also included as keywords of usability.
- **Capability**: First, the alternatives of the word 'capability' should be included in the list, such as ability, function and feature, etc. Then variants of these words including capability should also be added, such as capable, able and functionality, etc. Adjectives and adverbs that one would use in proposing a capability user request are also keywords, such as great, nice, possible and certain, etc. At last, we include the verbs that one would mention when proposing a new function request, such as add, allow, enable, customize and implement, etc.
- **System Interface**: According to the definition, there should be communications between the current system and another one. So, words implying a communication should be added firstly, such as API, plugin, interface, return, callback, protocol, etc., as well as words for describing an API, such as source, global, import, package and library, etc. Protocols (except "http") for communications between systems are also used as keywords, such as file://, ftp://, sftp://, etc.

So far, we have only introduced the method for generating cross-projects reusable keywords of each type of requirements and give examples of them. These keywords are just a part of those used in our classifiers. However, for different projects, there must be words representing project specific knowledge that can be used as keywords. In theory, by adding those words, the accuracy of classification should be improved.

Regardless of whether user requests are collected through specialized platform in industry projects or through user forums in open source projects, there are simple descriptions of the projects or the software that tell users the basic expectations of the product and the project-specific background knowledge. In open source communities, e.g. *github.com* and *sourceforge.net*, there are even summaries of the classified features listed on the homepage of the project. So, by reading the project description and the feature list, we can derive project-specific requirement keywords. Another way is to make use of the large number of unlabeled and yet to be classified user requests using the NLP technique *Word2vec*.[3] *Word2vec* calculates a vector for each word to represent its meaning using the words that appear in its contexts. Two words having similar vectors are supposed to be similar to each other in the given input documents. As we already have non-project-specific keywords of each type, we could use them as seeds to find more project-specific keywords that are similar to them via Word2vec. There are two concrete methods for finding similar keywords:

[3] Word2vec is a two-layer neural net that processes input text corpus and its make feature vectors for words in that corpus. Refer to page: https://deeplearning4j.org/word2vec.

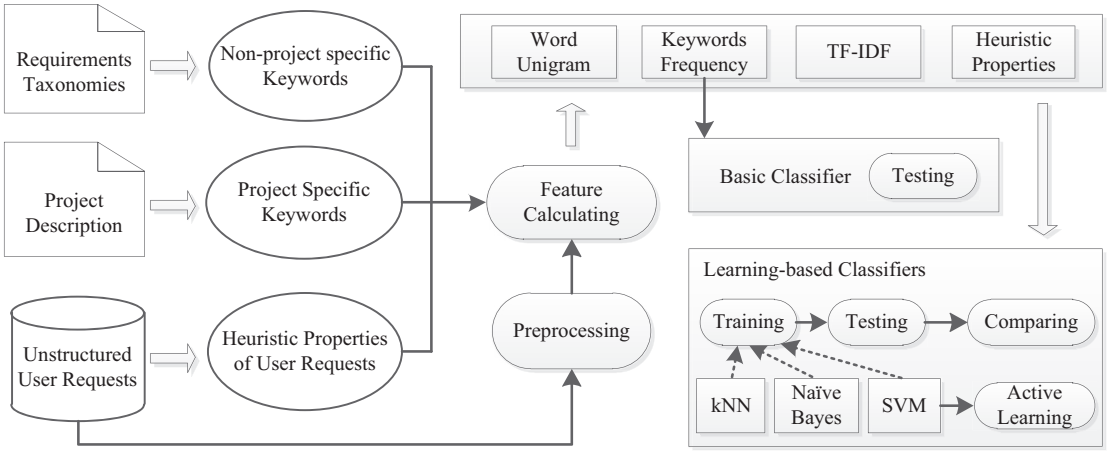


Fig. 1. Overview of the framework for building the proposed classification approach.

(1) Words similar to each word in each type: use each word of each type as a single seed, calculate its similarities with each of the other words, and retrieve the *k* most similar words.
(2) Words similar to the average of words in each type: calculate the average vectors of all keywords in each type, find the *k* words whose vectors are most similar to the average one, or find the most similar one each time, then add it and re-calculate the average vectors, until *k* words are added.

Although Word2vec will take advantage of the large number of unlabeled user requests and reduce the manual efforts in labeling keywords, it is just a complementary way to find project-specific keywords. Besides, each of these potential keywords should be manually inspected and added to the list only if it makes intuitive sense. After introducing three projects used in our experiments, we would present examples of project-specific keywords for them.

4. Approach

Section 4.1 presents an overview of the framework we used in finding the approach for user requests classification. Section 4.2 describes the features we choose to represent requests in machine-learning-based classifiers. Section 4.3 introduces both a simple non-learning-based classifier and the learning-based classifiers by illustrating four different machine-learning algorithms to be used.

4.1. Overview

As shown in Fig. 1, the classification approach starts from manually generating a list of reusable keywords (i.e., non-project specific keywords) of requirements obtained by analyzing software requirements taxonomies. Considering domain differences between projects, we also retrieve project-specific keywords from project description and unlabeled user requests. Besides keywords, a group of heuristic properties of requests summarized from informal natural language are also used as features in learning-based classifiers. In order to reduce the influence of different forms of a word, we transform words into their basic lemma by using lemmatization in preprocessing. All derived keywords are also lemmatized. For training and evaluating learning-based classifiers, we manually label user requests according to the class definitions presented in Section 3. In order to find the learning-based approach with the highest classification accuracy, we train, test and compare different classifiers using different combinations of representations of user requests and machine-learning algorithms. Besides keywords and heuristic properties, we also employ techniques in natural language processing to represent request documents for machine-learning algorithms, such as word unigrams and TF-IDF, which are widely used in document classification research. Supervised machine learning algorithms such as k-NN, Naïve Bayes and Support Vector Machine are used in building classifiers. To highlight the advantage of learning-based classification approaches, we build a simple non-learning-based classifier with number of keywords as a baseline.

4.2. Representations of user requests

As shown in the first three columns of Table 2, the input of a classification approach is a user request consisting of request records, each of which contains a textual “description” and an optional “summary”. In training learning-based classifiers, in order to create input for machine learning algorithms, we need to transform these request items into other formats. We use four kinds of representations of textual requests, as described below.

4.2.1. Word unigram and keyword unigram

In the word unigram format, each request record is represented as a binary-valued vector of features, where each feature corresponds to a distinct word that appears in the training set. Specifically, if the feature corresponds to a word that appears in the record, its value is 1; otherwise, its value is 0. In other words, the vector indicates the presence or absence of a word in the corresponding record. Also note that we did not remove stop words from the vector, as preliminary experiments indicated that results deteriorated slightly with their removal. This makes sense because stop words play an important role in textual expressions of people’s ideas. Taking stopwords *but* and *not* in the 4th request in Table 2 as examples, the absence of expected appearance that indicates the unreliability of the software would not be expressed without them. Besides, you would feel strongly how badly the security situation is while reading stopwords *anyone can* in the 5th request in Table 2. So, in all representations of user requests, we do not remove stop words. The reason for trying word unigrams is that it is a widely used technique in document classification. Classifiers based solely on word unigrams will also be used in the comparison among all classifiers. Since we have already identified all keywords of each type of requirements, we can also use only keywords instead of all distinct words appear in the training set to calculate the features, which we refer to as keyword unigrams.

4.2.2. Word frequency and keyword frequency

In the word frequency format, each request record is also represented as a binary-valued vector of features, where each feature also corresponds to a distinct word that appears in the training set. The difference is that the value of each feature is its frequency of occurrence in the record. However, in preliminary experiments, the performance of using the frequency of a distinct word in the training set as a single feature is not as high as only using the frequency of a keyword of each type as a feature. So, we will only present experiment results of using keyword frequency in the evaluation section.

4.2.3. TF-IDF

Both word unigram and word frequency are types of the basic representation of a document called bag of words (BOWs). TF-IDF is another common type of BOW: it is a more complex alternative to word frequency. Typically, the TF-IDF weight is computed as the product of two terms: the first computes the normalized Term Frequency (TF), i.e., the number of times a word appears in a document; the second term is the Inverse Document Frequency (IDF), which measures how important a term is. Since documents differ in length, it is possible that a term would appear many more times in a long document than in the shorter ones. Thus, the term frequency is often divided by the total number of words in the document as a way of normalization. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "the", may appear a lot of times but have little importance. Thus we need to weight down the frequent terms while scale up the rare ones. Here is the formula for calculating TF-IDF of a word w in document d:

$$\mathrm{TF-IDF}(w,d) = \mathrm{TF}(w,d) * \mathrm{IDF}(w), \text{ where} \quad (1)$$

$$\mathrm{TF}(w,d) = (\text{frequency of } w \text{ in } d)/(\text{number of words in } d), \text{ and} \quad (2)$$

$$\mathrm{IDF}(w) = \log\_e(\text{number of documents in the corpus}/ \times \text{number of documents with } w \text{ in it}) \quad (3)$$

4.2.4. Heuristic property of user request

Each user request can be split into different parts and there are certain parts that strongly imply the potential classes of the request. For example, in usability requests, there may be a part for expressing the convenience that the proposed feature would bring to users or the inconvenience that users would suffer from without this feature, and the convenience and inconvenience descriptions both strongly imply that this is a usability request. We name different parts of a request that imply its classes *Heuristic Property* (HP). Since a user request may have different HP implying different request types and a single HP may imply more than one request type, we will explore the relationship among HP and request classes by using HP as a kind of feature to represent the user requests in learning-based classifiers.

The heuristic property of requests is similar to the *Linguistic Pattern* proposed by Toro et al. (1999) and the Level 2 (L-2) in Requirements Classification for Natural Language (RCNL) ontology proposed by Vlas and Robinson (2012). Linguistic Patterns are most often used words, phrases or sentences in natural language requirements descriptions that can be parameterized and integrated into requirements templates. L-2 in the RCNL ontology defines expressions of contexts that can indicate a requirement. However, there are also significant differences between HP and the others. Linguistic Patterns are proposed to improve requirements elicitation and expression in the early phase in requirements engineering and they are pre-defined by experts, while HPs are derived from existing user requests. Although specific expressions in L-2 of RCNL ontology are also extracted from user requests, they are used to retrieve all types of requests while the HPs defined by us are for figuring out the exact class of a request. Besides, concrete concerns in L-2 of the RCNL ontology, which are contexts of belief, certainty, necessity, preference, qualifier, quantifier, qualifying phrase, are different from those of HPs.

We define seven heuristic properties for user requests. Their names and definitions are:

(1) **System Capability**: The core functions of the system that the owner of the request wants to be implemented.
(2) **Rationale:** The benefits of implementing this request, or unexpected or negative things that would happen if this request were not implemented.
(3) **Related Existing Capabilities**: The existing function or component that is related to the proposed request.
(4) **Expected or Unexpected Behavior**: What the system should or should not behave according to the request.
(5) **Context**: The conditions under which the expected or unexpected behavior should or should not appear.
(6) **Implementation Instructions**: It concerns how to implement the request function, for example, steps that the developers should follow to develop the requested function.
(7) **Examples**: Examples for illustrating anything proposed in the request.

While used as features for training learning based classifiers, values for each HP can only be '0′ or '1′. If there is no sentence in the request corresponding to a HP, the value for that HP is '0′ otherwise it is '1′. We do not retrieve words that represent the content of a HP to check if that HP exists in one user request. Just like *Linguistic Pattern* proposed by Toro et al. (1999), we define patterns of common expressions for each HP. If one request matches any expression belonging to a HP, then the value for that HP in this request is set to '1′ and otherwise '0′. In other words, we only need to check the existence of common expressions that imply the existence of HP in each user request to calculate values for HPs. Table 3 shows some example patterns of expressions of different HPs.

Besides values for HPs, we also calculate the other four values to be used as features of user requests. These values are motivated by natural language processing approaches using Conditional Random Fields (CRF) (Jiao et al., 2006), where index of words or sentences, and number of sentences are often used as features of data instances. In the remaining parts of this paper, the four values are also viewed as a part of HPs. The four values are:

(1) ***Index*** of sentence introducing **System Capability**: if there is no such sentence, the value is '0′; if it is the first sentence, the value is '1′; otherwise, it is '2′.
(2) ***Times*** that words refer to human beings appear: words like I, you, we, user, etc.
(3) ***Times*** that words refer to system or system components appear: words represent *the name of the system*; words like program, software, system, database, etc.; words represent the *name of a UI component*, etc.
(4) ***Number*** of sentences that do not contain any one of the aforementioned HPs in the request.

All these HPs' influences on classification accuracy of user request are investigated in Section 5.2.

4.3. Classifiers

We use the keyword-based requirements classification method proposed by Cleland-Huang (2009) as a basic classifier that is used for comparing with the learning-based ones.

Table 3
Example patterns of common expressions of different Heuristic properties.

System Capability	Sentence starts with a *verb* or the word *Please* followed by a *verb*; Interrogative sentence starts with *Can/Will you/it, How/What about, Why not* or *Is it possible* etc.; Declarative sentence starts with *It will be adjective to/if* or *I would like to/appreciate/propose/want/recommend/am looking forward* etc.
Rationale	Sentence contains *so that, I think, that way, since, because, as…when, then…can/be able to*, or *the reason*, etc.
Related Existing Capability	Sentence contains *current, currently, sometimes, cases, now, every time, at the moment*, or *presently, unfortunately*, etc.
Expected or Unexpected Behavior	Sentence contains *should, should not, however…not, but…not, in addition to, rather than, instead of, even…not*, etc.
Context	Sentence contains *when, if, while, after, before*, etc.
Implementation Instructions	Sentences in a raw and start with *First, Second, Next* or *Then, Last*, etc. respectively.
Examples	Sentence contains *for example*, e.g., i.e., *like/as in, same as, different from, similar to, such as*, etc.

4.3.1. Basic classifier

First, the frequencies of keywords of different types of requests are calculated for each request. In Cleland-Huang et al. (2009), a request will be classified as a candidate of all classes if it contains one or more of their keywords. So, a request may be classified into more than one category, and it is the same with the classification method proposed in Vlas and Robinson (2012), which is based on the Requirement Classification Natural Language Ontology. However, the goal of our approach is to determine only one class for each request. So, we will classify the request into the class that the request contains most keywords of it. If there is more than one class that the request should be classified, we break ties by selecting the class that has less request instances in the entire dataset.

4.3.2. Learning-Based classifiers

After the feature vectors of user requests are calculated, we apply off-the-shelf machine learning algorithms to train classifiers that can be used to classify unseen records to one of the predefined request categories. In other learning-based structured or unstructured requirements classification research (Casamayor et al., 2010; Zhang et al., 2011; Riaz et al., 2014), the most used machine learning algorithms are k-nearest neighbor (k-NN), Naïve Bayes (NB) and Support Vector Machine (SVM). We will evaluate all these learning algorithms by training the corresponding learned classifiers.

For SVM and NB, we try both the binary (B) and the multiclass (M) versions. While using binary classifiers, we adopt the one-versus-others training scheme, where we train one SVM/NB classifier for each class. For instance, we train one classifier for determining whether a record belongs to Capability, another classifier for determining whether a record belongs to Usability, etc. In essence, each classifier represents exactly one requirement class, and the number of SVM/NB classifiers we train is equal to the number of requirement classes. The training set for training each of these classifiers, of course, is different from each other. Specifically, to train a classifier for determining whether a record belongs to class *i*, we take the pre-processed training set and assign the class value of each training instance as follows. If the training instance belongs to class *i*, its class value is 1; otherwise, its class value is 0. While classifying a test record, each classifier will calculate the posterior probability of the record belonging to the predicted class. As a result, we assign to a request record the class whose classifier returns the highest probability value among the set of values returned by all the classifiers. The process of training multiclass versions of SVM and NB is simpler. Only one SVM/NB classifier needs to be trained with the prepared training set where each request record is labeled with its correct class. In testing, each test record is delivered to the only one classifier and it will return its predicted class.

In order to reduce the number of required labeled requests for training a classifier, we employ *active learning* (Cohn et al. 1994) to see the learning curve by increasing the number of requests in training set. Active learning is a semi-supervised machine learning strategy to reduce human efforts in labeling training data (i.e., label as fewer instances as possible) while still achieve a high accuracy classifier.

5. Evaluation

In this section, we evaluate the contribution made in this paper by answering the five corresponding research questions with experiments results. Section 5.1 introduces data sets, experiments environment and evaluation metrics. Section 5.2 presents results of experiments and answers research questions. Section 5.3 compares our research with related ones and discusses the implications of this study.

5.1. Data collection

We collect user requests of open source projects from sourceforge.net, whose user base consists of both software developers and ordinary software consumers. The major criteria for selecting projects are:

(1) each project should have a high ranking in projects list provided by sourceforge.net,
(2) each project should have more than 1000 feature requests,
(3) all projects should focus on different quality perspectives, and
(4) all projects should belong to different categories.

By investigating other works in requirements engineering, we find that if we want to make the proposed approach convincible, we need to at least collect experiment results from three projects. Eventually, three projects we chose from sourceforge.net are *KeePass, Mumble* and *Winmerge*. Here are brief introductions of them:

(1) **KeePass** (Category: Business & Enterprise/Office/Business): a portable password manager for storing all passwords of different accounts. By using KeePass, one does not have to use the same password over and over to avoid password management of a highly risky practice. One will never need to worry that if someone gains her social website's password, he can also log into her email accounts or even bank accounts.
(2) **Mumble** (Category: Communication/Internet Phone): a low-latency but high quality voice chat software primarily intended for using while playing online games. It provides services not only for game players, but also for server maintainers and software developers. One can download a Mumble client version and connect to a server or download a Mumble server version and setup in one's own network. If one is interested in software developing, he or she can also download source code of Mumble, implement his or her creative features, and build a new version. Users of Mumble are normal game players, system maintainers and software developers.
(3) **WinMerge** (Category: System Administration/Storage/File Management): a differencing and merging tool for both folders and files, presenting differences in a visual text for-

Table 4
Examples of project-specific keywords of each projects.

Project	*Security*	*Reliability*	*Performance*	*Lifecycle*	*Usability*	*Capability*	*System Interface*
KeePass	steal, authentication, keylogger, guess	strong, powerful, lock, require	separate, run, sometimes, start	client, applet, detect, create	note, header, column, confuse	store, database, merge, expiration	eWallet, PassKeeper, Truecrypt, convert
Mumble	information, pseudo, keychain, repeat	Reconnect, acceptable, fuzzy, override	kbite, infinite, megabyte, bottleneck	Linux, ChromeOS, distribute, device	enable, push, extra, username	volume, channel, register, music	Everquest, Warrock, Warcraft, Arma
Winmerge	write, save, disable, mode	check, happen, ignore, change	wait, much, improve, huge	manual, stable, translator, register	vertical, option, multiple, group	differ, review, pattern, bookmark	TortoiseSVN, LibcURL, Putty, Subclipse

Table 5
Statistics of each dataset: percentage of user requests of each type.

Project	Percentage of user requests of each type in each project (1000 user requests for each project)						
	Security	*Reliability*	*Performance*	*Lifecycle*	*Usability*	*Capability*	*System Interface*
KeePass	10.1%	6.8%	1.9%	3.9%	34.3%	38.1%	4.9%
Mumble	2.4%	6.3%	3.5%	12.3%	33.2%	33.7%	8.6%
Winmerge	0.9%	7.4%	2%	5.7%	41%	40%	3%

mat that is easy to understand and handle. With this basic functionality, it is highly useful for determining what has changed between project versions, and then merging changes between different versions.

After crawling all feature requests, we mined project-specific keywords according to aforementioned approach and Table 4 are examples of project-specific keywords for each projects. The next step is to label data.

Considering the scales of requirements used in related research, such as Cleland-Huang et al. (2007) (i.e., 684 requirements in total) and Vlas and Robinson (2012) (i.e., the average number of feature requests of projects is 1035), we prepare 1000 user requests for each project in our data sets. Each request has more than ten words. The *start* and *end* dates of requests are: (1) KeePass from 2003–10 to 2012–03, (2) Mumble from 2005–09 to 2016–06, and (3) Winmerge from 2000–12 to 2016–03. Each request is chosen by one of the co-authors according to the principle that each request should talk about one single feature of the product. The requests that mentions several features would be either split or just drop out. At the same time, this co-author labeled all of these requests. Upon all requests are selected and labeled, the other three co-authors start to label them referring to definitions of request types. After all co-authors finished, our labeled lists are merged in two steps: (1) If three or four people label a request as LABEL1, then the label for that request is LABEL1, (2) A discussion is held to determine the request type based on labeled results.

Table 5 shows the statistics on percentages of each type of requests in eventual data sets of each project. Like in most software projects, the most proposed user requests of the three projects are about capability and usability. As shown in Table 5, the percentage of capability and usability requests are 72.4%, 66.9% and 81% in KeePass, Mumble and WinMerge respectively. Capability and usability are the majority classes and others are the minority classes. The main differences among these three projects lie in the distribution of minority classes.

5.2. Evaluation methodology and metrics

To compare the learning-based classifiers fairly, we use implementations of machine learning algorithms in *Weka*.[4] For each learning algorithm, we obtain its performance using five-fold cross-validation experiments on each dataset. Specifically, we first randomly partition each dataset into five equal-sized subsets (or folds). Then, in each fold experiment, we reserve one of the five folds for testing and use the remaining four folds as the training set. We repeat this five times, each time using a different fold as a test set. Finally, we average the result obtained over the 5-folds.

We employ three evaluation metrics. First, we report overall accuracy, which is the percentage of records in the test set correctly classified by a request classifier. Second, we compute the precision and recall for each request category. Given a request category c, its recall is the percentage of request records belonging to c that are correctly classified by the classifier; and its precision is the percentage of records classified by the system as c that indeed belong to c. The F-measure for each request category is computed as

$$F - measure = 2 * precision * recall/(precision + recall) \quad (4)$$

F-measure represents the balance between precision and recall. The higher the F-measure of a category is, the better the performance of the classifier on this category is.

5.3. Experiments and results

This section presents the experiment results for answering research questions. Description after each research question consists of two subsections, *Method* and *Results*. For the sake of convenience, abbreviations of common terms are used in this section. Specifically, *WU* stands for word unigrams, *KU* stands for keyword unigrams, *KF* stands for keyword frequency, *HP* stands for heuristic properties of user requests, *NB-B* and *NB-M* stand for the binary and multi-class versions of Naïve Bayes respectively, and *SVM-B* and *SVM-M* stand for the binary and multi-class versions of Support Vector Machine respectively.

RQ1. *How well can those user requests be classified by only using widely used simple text features as input of machine learning algorithms,* e.g., *word unigram and TF-IDF?*

Method. Train and test classifiers for each data set on features of *WU* and *TFIDF* with different machine learning algorithms, namely *k-NN, NB-B, NB-M, SVM-B* and *SVM-M*.

Results. The highest classification accuracy is achieved by using *TFIDF* as features for KeePass, Mumble and Winmerge, and their corresponding accuracies are 68.1%, 67.5% and 72.2% respectively. Different from the other two projects, it is SVM-M algorithm

[4] Weka is a collection of machine learning algorithms for data mining tasks and it can be found in: http://www.cs.waikato.ac.nz/ml/weka/downloading.html.

Table 6
Overall accuracies of learning-based classifiers using word unigram and TFIDF as features.

Algorithm	KeePass		Mumble		Winmerge	
	WU	*TFIDF*	*WU*	*TFIDF*	*WU*	*TFIDF*
kNN (%)	39.4	39.1	35.3	35.7	48.6	44.8
NB-B (%)	63.4	45.1	61.6	44.4	69.7	52.7
NB-M (%)	63.7	46.2	62.9	48.6	68.4	56.4
SVM-B (%)	67.5	*68.1*	62.4	*67.5*	71.2	70.8
SVM-M (%)	65.8	67.2	63.3	66.1	70.7	*72.2*

Table 7
Results of basic and learning-based classifiers. ('*KU*' for keyword unigram, '*KF*' for keyword frequency, '*NPS*' and '*ALL*' for 'non-project-specific' and 'both non-project and project specific' keywords, '↑' represents '*increase*').

Projects	Keyword type	Basic (%)	k-NN(%)		NB-B(%)		NB-M(%)		SVM-B(%)		SVM-M(%)	
			KU	*KF*	*KU*	*KF*	*KU*	*KF*	*KU*	*KF*	*KU*	*KF*
KeePass	*NPS*	51.6	43.6	53.3	62.8	61.9	63.0	61.7	64.6	61.7	62.8	64.5
	ALL ↑	52.2	46.2	54.6	65.2	65.7	64.3	65.6	66.3	70.2	65.3	69.5
Mumble	*NPS*	49.9	41.3	49.3	60.5	58.8	60.6	58.5	62.4	62.1	61.3	61.7
	ALL ↑	67.2	48.8	65.7	67.1	70.7	67.8	70.7	66.3	71.4	65.9	72.7
Winmerge	*NPS*	53.9	52.5	62.1	67.2	67.8	67.6	67.9	69.0	68.1	67.0	70.0
	ALL ↑	66.7	52.6	67.7	69.9	73.9	69.3	73.2	73.3	76.2	70.8	74.3

Table 8
Results of training with TFIDF, all keywords, and TFIDF plus all keywords separately. (only KF of keywords is used, '*ALL*' for 'both non-project and project specific' keywords, '↑' represents '*increase*').

Algorithm	KeePass			Mumble			Winmerge		
	TFIDF	*ALL*↑	*TFIDF+ALL*	*TFIDF*	*ALL*↑	*TFIDF+ALL*	*TFIDF*	*ALL*↑	*TFIDF+ALL*
kNN (%)	39.1	54.6	45.8	35.7	65.7	38.2	44.8	67.7	49.6
NB-B (%)	45.1	65.7	46.3	44.4	70.7	49.3	52.7	73.9	57.1
NB-M (%)	46.2	65.6	57.2	48.6	70.7	59.4	56.4	73.2	66.2
SVM-B (%)	68.1	70.2	74.6↑	67.5	71.4	74.9↑	70.8	76.2	78.4↑
SVM-M (%)	67.2	69.5	74.8↑	66.1	72.7	74.2↑	72.2	74.3	78.1↑

achieving a better accuracy than SVM-B in Winmerge. Table 6 shows the details of the results.

RQ2. *Will the classification accuracy of entire data sets as well as those of minority classes be improved by using keywords-based features?*

Method. Use non-project-specific keywords to build basic and learning-based classifiers separately. Then, add project- specific ones to see if the accuracies are improved. At last, using TFIDF with all keywords as features together in training learning-based classifiers to see whether the accuracies are improved. Investigate concrete precisions (P) and recalls (R) of minority classes in of different classifiers.

Results. First, after adding project-specific keywords to keywords lists, the accuracies of both basic and learning-based classifiers will be improved (Table 7). We also find that using keyword frequency will achieve better classification accuracies than keyword unigram. Second, only using keywords as features will improve the accuracies, but use TFIDF with keywords will improve the accuracies more significantly while training with SVM (Table 8, only *KF* is used). At last, it shows that keywords-based features help achieve better performance on classifying minority classes (Table 9, train with SVM-B).

RQ3. *To what extent are the proposed non-project-specific keywords accepted by others? Are they more effective than those proposed by others?*

Method. We investigate fifty students' attitudes on our keyword lists in Software Institute of Nanjing University, China. We also collect extra keywords that they want to add to our lists. Students are junior and senior undergraduates, and graduates of the first and second year. At the beginning, we introduce definitions of the seven types of user requests defined by us as well as the documents referred by us to retrieve keywords to all students. Then, we show them keywords found by us and let them mark the words that they agree with. At the same time, we let them propose new keywords that they think are missed. Afterwards, we gather all new keywords proposed by students and again let each student mark those they agree with. Eventually, we make statistics on the survey results. According to students' feedbacks, we made changes on our list and conduct new experiments with them. At last, we compared results of new experiments with former ones.

Results. Table 10 shows that most keywords of each class proposed by us are accepted by more than 40% students. Fig. 2 tells that for all data sets, the non-project-specific keywords proposed by us really perform better than others. In Table 10, the value '29′ in the green cell means that there are twenty-nine out of all thirty-one *Security* keywords proposed by us also agreed by more than or equal to 40%* 50 (i.e., 20) students; the value '2′ in the red cell means two words newly proposed by students are accepted by 15 to 20 students and the words are "throughput" and "response" . Fig. 2 compares classification accuracies among using our keywords and revised ones. There are three kinds of revised keywords lists based on Table 10 and they are: (1) remove keywords in U (i.e., proposed by us) where $PoS < 20\%$, e.g., *accident, but, not, instance* and *homepage*, (2) all words in U adding those in S (i.e., newly proposed by students) where $30 \leqq PoS < 40\%$, e.g., *backup, steal, tolerance, throughput, response, migrate, beautiful, tap* and *touch*, (3) add words in S where $20\% \leqq PoS < 30\%$ in lists of (2), e.g., *down, latency, rate, memory, upgrade, smooth, style, solve, implement, connection, supply* and *service*.

RQ4. *Which machine learning algorithm that widely used in requirements classification do best here, k-nearest neighbor, Naïve Bayes or Support Vector Machine?*

Method. Apply On-way ANOVA to classification accuracies derived from classifiers trained by different learning algorithms.

Table 9
Comparison of precisions (P) and recalls (R) of minority classes derived by SVM-B. (only KF of keywords is used, '*ALL*' for 'both non-project and project specific' keywords, '↑' represents '*increase*').

Projects	Features	*Security*		*Reliability*		*Performance*		*Lifecycle*		*System Interface*	
		P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
KeePass	**TFIDF**	65.3	50.4	58.3	20.5	71.4	26.3	86.6	66.6	75	48.9
	TFIDF+ALL	75	62.3	64	47.1	63.6	36.8	94.3	84.6	80	65.3
	↑of F-measure	11.2		23.8		8.2		13.8		12.6	
Mumble	**TFIDF**	0	0	74.2	41.2	80.7	60	70.4	56	82	63.9
	TFIDF+ALL	82.3	58.3	80	69.8	88.9	68.5	80.1	72.3	77.8	73.2
	↑of F-measure	68.2		21.5		8.6		13.6		3.6	
Winmerge	**TFIDF**	0	0	58.1	24.3	63.6	35	70.2	57.9	81.8	60
	TFIDF+ALL	87.5	77.8	72.9	47.3	94.1	80	75.5	64.9	86.2	83.3
	↑of F-measure	82.4		23.1		41.3		6.4		15.5	

Table 10
Statistics on acceptance of each type of keywords in this paper (*U*) and proposed by students (*S*). (*PoS* means percentage of students; in "()" are concrete keywords).

Types	Total Number		40%≦*PoS*		30%≦*PoS*<40%		20%≦*PoS*<30%		*PoS*<20%	
	in *U*	in *S*	in *U*	in *S*	in *U*	in *S*	in *U*	in *S*	in *U*	in *S*
Security	31	31	29	0	1	2(backup, steal)	0	0	1(accident)	29
Reliability	29	32	25	0	0	2(backup, tolerance)	2(current, none)	1(down)	2(but, not)	29
Performance	30	17	24	0	3	2(throughput, response)	2(large, occupy)	3(latency, rate, memory)	1(instance)	12
Lifecycle	44	49	33	0	8	1(migrate)	2(binary, download)	1(upgrade)	1(homepage)	47
Usability	76	89	75	0	0	3(beautiful, tap, touch)	1(big)	2(smooth, style)	0	84
Capability	19	54	13	0	4	0	2(nice, please)	2(solve, implement)	0	52
System Interface	21	50	20	0	1	0	0	3(connection, supply, service)	0	47

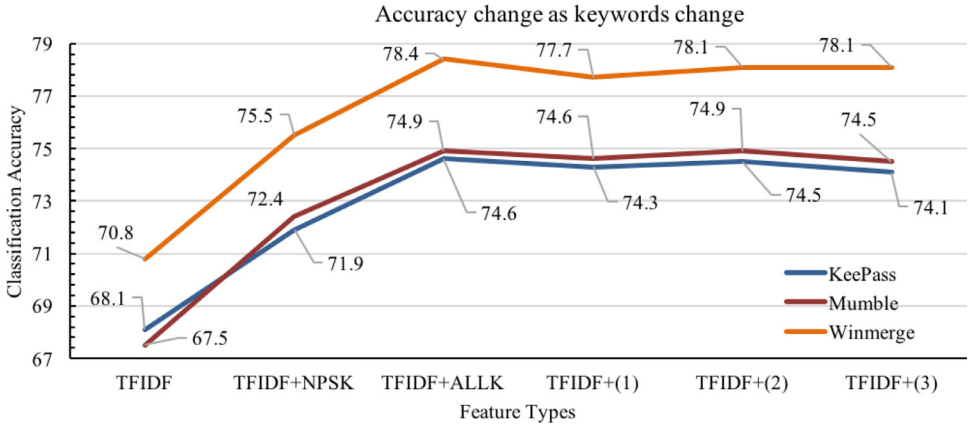


Fig. 2. Comparisons of accuracies among classifiers derived by training with different features with different lists of keywords (NPSK means non-project-specific keywords, ALLK means all of non-project-specific and project specific keywords).

Compared algorithms are: k-NN, NB-B, NB-M, SVM-B and SVM-M. Each algorithm has ten accuracies on each project and they are trained on features of: *WU, TFIDF, KU-NPS, KU-ALL, KF-NPS, KF-ALL, TFIDF* plus *KU-NPS, TFIDF* plus *KU-ALL, TFIDF* plus *KF-NPS*, and *TFIDF* plus *KF-ALL*. So, there are 30 data points for each compared algorithm. First, we test Homogeneity of Variances of all five groups of accuracies. Then choose one test method to conduct Post Hoc Tests. The hypothesis of ANOVA is that there are no significant differences among classification accuracies of classifiers trained by different learning algorithms.

Results. The answer to this question is that SVM does best in classifying user requests in crowdsourcing scenario. The significant value of Homogeneity of Variances Test is 0.032 and it is less than 0.05. So, we apply one Nonparametric Test method, i.e., Tamhane, in Post Hoc Tests. Table 11 shows that binary version of SVM achieves significantly higher classification accuracies than k-NN and NB algorithms. However, there is no big differences between binary and multi-class version of SVM.

RQ5. *Do the proposed Heuristic Properties of user requests influence the classification results? How do they influence?*

Table 11
Results of ANOVA among different learning algorithms: Post Hoc Tests using Tamhane.

Learning algorithm	NB-B		NB-M		SVM-B		SVM-M	
	Mean Diff.	*Sig.*	*Mean Diff.*	*Sig.*	*Mean Diff.*	*Sig.*	*Mean Diff.*	*Sig.*
kNN	-12.70952	0.000	-14.52381	0.000	-20.60952	0.000	-20.20000	0.000
NB-B			-1.81429	0.998	-7.90000	0.011	-7.49048	0.018
NB-M					-6.08571	0.019	-5.67619	0.034
SVM-B							0.40952	1.000

Table 12
Comparisons among classification accuracies of HP related classifiers and non-HP related ones. (only keyword frequencies of keywords are used; 'HP' for heuristic properties, 'TALL' for TFIDF plus all keywords, 'TAHP' for TALL plus HP).

Algorithm	KeePass				Mumble				Winmerge			
	HP	*TFIDF*	*TALL*	*TAHP*	*HP*	*TFIDF*	*TALL*	*TAHP*	*HP*	*TFIDF*	*TALL*	*TAHP*
SVM-B (%)	64	68.1	74.6	75.9	50.3	67.5	74.9	75.5	59.6	70.8	78.4	79.6
SVM-M (%)	64.3	67.2	74.8	75.6	52	66.1	74.2	73.8	58.8	72.2	78.1	78.4

Method. Add heuristic properties to feature set and compare the classification accuracies with others.

Results. Using Heuristic Properties as the single type of features cannot achieve as high classification accuracies as TFIDF or all keywords can. But, after adding Heuristic Properties to TFIDF and keywords, most of the classification accuracies are improved again (Table 12, only SVM learning algorithms are used).

RQ6. *Can keywords-based features help reduce the size of training sets to achieve the same accuracy derived by using widely used features?*

Method. Apply active learning using SVM-M on keywords-based features and TFIDF separately, and draw the learning curves for both of them as the training set expands. The feature, whose learning curve converges more quickly than the other, do better in reduce the size of training sets that needed for an acceptable classification accuracy. There are four steps in applying active learning:

(1) train a classifier with labeled training data you have,
(2) find the instance that the derived classifier has *least confidence* to assign it among all unlabeled instances you have,
(3) label this instance manually and add it to the training data,
(4) repeat step 1 to 3 until the classifier achieve an acceptable accuracy on testing data.

Step 2 defines the approach to choose next instance to label in active learning. There are different ways to choose next instance according to different understanding of 'least confidence'. We try several ways in our experiments and here introduce three widely adopted ones, namely ***MinMax***, ***MinGap*** and ***Random***. First, for each unlabeled instance, use the current classifier to classify it and *probabilities* for different classes would be derived. Then, conduct different ways to choose next instance to label:

(1) ***MinMax***: First, use the biggest probability to represent the *confidence* of the classifier to correctly classify the unlabeled instance. Then, among all unlabeled instances, choose the one with the lowest confidence as the next instance to label.
(2) ***MinGap***: First, use the gap between the top two probabilities to represent the confidence of the classifier to correctly classify the unlabeled instance. Then, choose the instance with the lowest confidence as the next one to label.
(3) ***Random***: Randomly choose an unlabeled instance as the next one to label.

Results. The answer is that keywords-based features will help reduce the size of needed labeled data to train classifiers with high accuracies, comparing with TFIDF features. Fig. 3 shows the active learning curves of three projects using different approaches to select next instance to label. In all projects, it reads that keywords-based features let the curves converge more quickly than TFIDF. However, different approaches for selecting next instances almost lead to learning curves of a same converging speed.

5.4. Discussion

We discuss the experimental results and implications of this study, as well as comparisons between our research and related ones in this section.

5.4.1. Meaning of results and implications

Overall, the experimental results are encouraging. These results prove that the non-project-specific keywords found by us are helpful in improving classification accuracies comparing with those derived by using widely used text features in NLP research domain, i.e., word unigram and TFIDF. Besides, after adding project-specific keywords, the performances of classifiers are further improved. Using *keyword frequency* (KF) to encode keywords-based features will achieve better performance than *keyword unigram* (KU). Besides keywords, the proposed Heuristic Properties of user requests are also effective in improving classification accuracies while using together with other text features.

Since there are much less user requests of minority types than others, classifiers trained with simple text features performs bad in classifying them. To the best of our knowledge, it is a big challenge in machine learning applications to achieve high accuracies of minority classes with unbalanced data sets. However, details of precisions and recalls (Table 9) of all minority classes derived in our approach show that using both non-project-specific and project-specific keywords proposed in this paper as training features to some extent help in overcoming bad classification results on minority classes. Table 13 shows the precisions and recalls of all classes while use SVM-B to train on combination features of TFIDF, frequency of all keywords and heuristic properties. Considering these values of precisions, recalls and F-measures, we cannot even tell exactly which are minority classes without information about combination of data sets showing in Table 5.

The experimental results also show that the machine learning algorithm SVM is more suitable in classifying user requests than other common used ones, i.e., k-NN and Naïve Bayes. Training classifiers with the binary version of SVM and using features encoding all keywords, heuristic properties, and TFIDF achieves classification accuracies above 75% for all three datasets (Table 12), which are really encouraging.

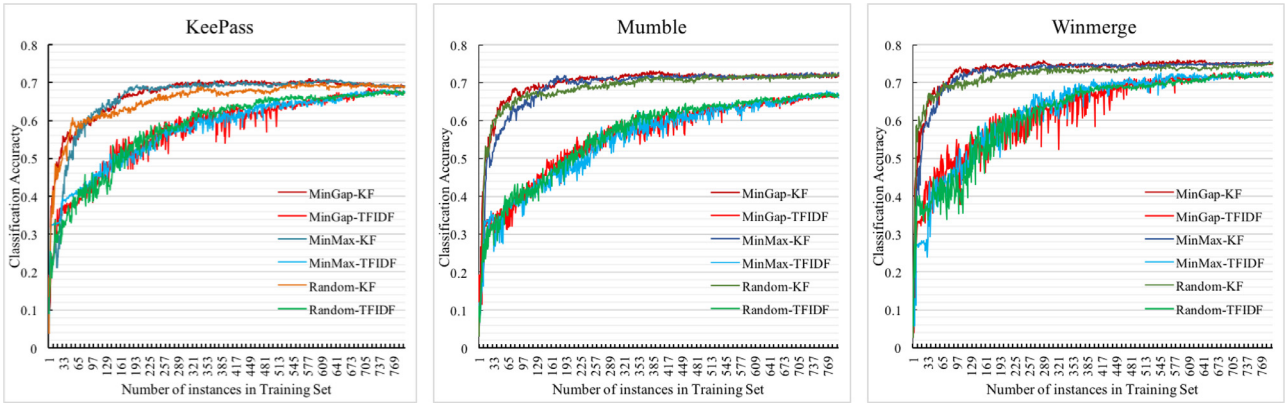


Fig. 3. Learning curves of applying Active Learning with different instance selection method.

Table 13
Statistics on precision (P), recall (R) and F-measure while training with SVM-B on all features.

Dataset	*Security*		*Reliability*		*Performance*		*Lifecycle*		*Usability*		*Capability*		*System Interface*	
	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
KeePass	86.0	73.3	66.7	50.0	100	31.6	88.9	82.1	74.6	77.0	73.4	81.9	88.1	75.5
Mumble	77.3	70.8	86.3	69.8	83.3	57.1	79.1	70.7	77.9	78.6	69.8	79.5	78.4	67.4
WinMerge	80.0	44.4	68.6	47.3	93.8	75.0	78.7	64.9	80.6	83.9	79.1	84.3	85.7	80.0
Average	81.1	62.8	73.9	55.7	92.4	54.6	82.2	72.6	77.7	79.8	74.1	81.9	84.1	74.3
F-measure	70.8		63.5		68.6		77.1		78.8		77.8		78.9	

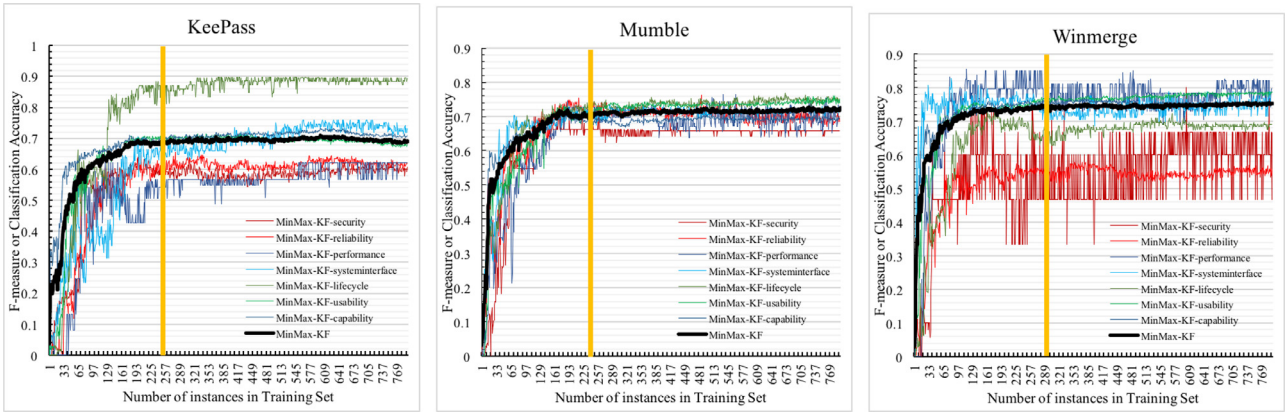


Fig. 4. Learning curves of F-measures of several minority classes while applying active learning. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Facing the large number of software artifacts, applying machine learning methods to automatically process them is strongly desired. However, the biggest obstacle of employing machine-learning methods is the enormous manual efforts involved in labeling those artifacts for training. So, the less labeled training data is required, the better. This is why we conduct experiments to answer ***RQ6***. Besides the overall classification accuracies, the precisions and recalls of minority classes should also be acceptable while using reduced number of labeled data. Fig. 4 shows comparisons among learning curves of overall classification accuracies and F-measures of all classes. The training algorithm is SVM-B and the features are frequencies of all keywords. The ***bold black*** curves are classification accuracies and other curves are F-measures of concrete classes. The ***bold yellow straight lines*** mark the converging speeds of all curves. It tells that F-measures of all classes converge similarly as the overall classification accuracies in all projects. (The red line in figure of *Winmerge* represents F-measure of class *Security*. Since the number of security requests in Winmerge is too few, i.e., 0.9%, the fluctuation of its F-measure is a bit wide.) Based on these results, we come to the conclusion: with active learning strategy, using keywords as features actually can reduce the needed labeled data for training an acceptable classifier. That means, using keywords-based features helps cut down manual efforts in labeling data.

5.4.2. Comparison with related works

In the analysis of the state of the art in automated requirements elicitation made by Meth et al. (2013), identification and categorization of requirements is positioned between *abstraction identification* tools and *requirements modeling* tools. This means requirements identification and categorization is a very important research direction to reduce the gap between natural language documents and structured software requirements. Among existing related works, research conducted by Cleland-Huang et al. (2006, 2007) and Vlas and Robinson (2012) are the most similar ones to ours. However, there still are significant differences between theirs and ours.

Cleland-Huang et al. (2006, 2007) proposed an iterative training framework based on the analyst's feedback for statements of

non-functional requirements identification and classification. The core component of their classifier is a set of requirements indicator terms (i.e., like k*eywords* in our paper) and the weights for each term belonging to each requirements class. The training process is to train weights of each term. The basic work is to mine indicator terms. Different from our method, they retrieve terms from training data directly. However, the effectiveness of these keywords relies on the number of instances labeled by humans as well as the feedback derived from manually analyzing. Besides, it is not mentioned that if the terms found by them from one project can also be used in other projects. It is true that their approach can be reused, but for different projects, different training set must be organized from scratch. Differently, outputs of our research are reusable non-project-specific keywords, methods for retrieving project-specific keywords and training strategies. This means there is no need to label data for training sets while applying our approach to a new project. That is why we say that our approach helps to reduce human efforts in user requests classification. In addition, they would classify one single requirement sentence into several different classes, while we only attach one type for each request. However, one of the most important factors that leads to differences between works of Cleland-Huang et al. and ours is the differences of application scenarios and data to be processed. They identify and classify requirements from requirements documents, such as specifications, but we classify product features proposed by ordinary users.

Vlas and Robinson (2012) proposed a rule-based user requests classification method. They also use user requests collected from sourceforge.net as experimental data sets. Their classification recalls are between 31% and 52% and precisions are between 27% and 45%. Considering that they allow classifying one request into more than one class and use as many as 23 classification criteria (we use only 7), that their classification results is much less than ours (i.e., recalls are between 54.6% and 81.9%, and precisions are between 73.9% and 92.4% on average for all data sets) makes sense. The seven types used in this paper are groups of existing requirements classes and we find that seven types are enough for most user requests processing scenarios according to its benefits to software requirements engineering. In addition, different from Vlas and Robinson (2012), who taking advantages of both linguistic patterns (i.e., L-patterns, natural language commonly used for describing requirements) and requirements patterns (i.e., R-patterns, generic requirements templates) in processing user requests, we propose heuristic properties of user requests but not patterns of requirements for classification of user requests. Because, there may be a doubt about whether patterns derived from requirements can be used to represent patterns of feature requests proposed by ordinary users.

6. Threats to validity

Threats to validity of this study are discussed in four aspects, namely, data collecting, types of user requests, keywords mining and approach applicability.

In data collecting, the categories of projects in sourceforge.net are used to control the variety of projects. But there is no existing matrix for calculating concrete values to measure the differences among projects. This is a potential threat to validity of the universality of the proposed approach. Besides, the manual user request classifications that needed for creating the training set and the test set, as well as the selection of project-specific keywords are subject to experimenter bias as they are done by the co-authors of this paper. Types of user request that used as classification targets in this paper are defined manually by grouping existing requirements classes. Different people may have different ideas on grouping these classes and naming each type. In addition, non-project-specific keywords for each type of user request are found manually from requirements related documents, which is subject to researcher bias. At last, since each user request is assigned to only one single class, it will achieve a higher overall classification accuracy if each input user request contains only one main topic corresponding to one type of software requirement. In other words, it is better to apply our approach to user requests where each one represents only one single feature of software or product. However, it is not to say that you must provide requests like we have selected in the experiments, otherwise our approach cannot work. To apply this approach in crowdsourcing requirements engineering projects, it is better to tell users to make sure each request item contains only one expectation of one product feature before collecting user requests. In our experiments, user requests are filtered manually according to this criterion before added into the ultimate data sets, which is also a potential thread to validity of the experiments results.

7. Conclusion and future work

This study contributes to the research and practice of user requests processing in crowd-sourcing software requirements engineering. Specifically, an automatic machine-learning-based method for classifying user requests on software or product features into classes of software requirements is proposed. This method can largely reduce human efforts in dealing with the large amount of user proposed requirements. Classifying requests collected from the crowd before the prototype developed will help to generate formatted software requirements specifications. If the requests are derived based on the experience of using a prototype or other versions of the product, e.g., open-source projects, classifying them would also help determine the releasing priority of proposed product features.

Since the proposed method is a machine-learning-based, natural language text classification approach, the main challenges are to find appropriate features to represent document items and machine learning algorithms to train classifiers. Our method uses TFIDF of words in corpus as basic document features. Then, propose non-project-specific keywords of different classes of software requirements for encoding text features. Besides, we develop some heuristic properties as features by interpreting the connections between the sentence structures of user requests and the definition of classes of software requirements. Another novel document feature used in our approach is encoding project-specific keywords. Keywords and heuristic properties derived from the definition of classes of software requirements are non-project-specific features, which means that the domain knowledge of the software projects would be omitted in classification if only those are used as document features. By adding project-specific keywords, classification accuracies are improved. Project-specific keywords can be derived from project introduction by labeling them manually or automatically from all user requests using the Word2vec technique. After the document features are prepared, we experimented with different machine learning algorithms to train classifiers and found that SVM did the best using the proposed features. The average accuracy of all datasets used in the experiments was 77%. Our method also performed well in handling imbalanced requests classes. For all classes, recall levels were between 54.6% and 81.9% and precision levels were between 73.9% and 92.4% on average. With active learning, our approach could reduce human efforts in labeling training instances while maintaining high classification performance.

We envisage future research in multiple directions. First, we will investigate other potential document features that would improve classification performance. Second, since experimental results show that it is better to reserve ‘stop words’ in user requests

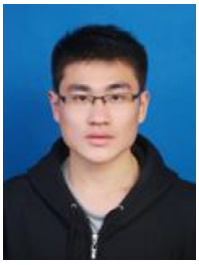
for classification, we would like to investigate why and try to find which 'stop words' have higher influence on classification accuracies in the future. Besides, how to automatically mine reusable and effective non-project-specific keywords from tremendous requirements related corpus is also a future work. Better reusable approaches for mining project-specific keywords from large number of unlabeled user requests are also worthy to research. As illustrated in Threads to Validity, it is better to apply our approach to classify user proposal that contains one single request on the product. So, how to split complex user proposals containing multiple requests on software features into simple ones that related to one single requested feature is also a future research direction.

Acknowledgments

This work was supported by the National Key R&D Program of China (2016YFC0800803), the National Natural Science Foundation, China (No. 61572162, 61572251, 61702144), the Natural Science Foundation of Jiangsu Province (No. BK20131277), U.S. National Science Foundation (NSF Awards CNS-1126747), the Fundamental Research Funds for the Central Universities. Jidong Ge is the corresponding author.

References

Adepetu, A., Ahmed, K.A., Al Abd, Y., Al Zaabi, A., Svetinovic, D., 2012. CrowdREquire: a requirements engineering crowdsourcing platform.
Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2015. Automated checking of conformance to requirements templates using natural language processing. IEEE Trans. Softw. Eng. 41 (10), 944–968.
Beck, K., 2000. Extreme programming explained: embrace change. addison-wesley professional.
Boehm, B.W., 1978. Characteristics of Software Quality, Vol. 1 North-Holland.
Boehm, B.W., 1981. Software Engineering Economics, Vol. 197. Prentice-hall, Englewood Cliffs (NJ).
Brabham, D.C., 2008. Crowdsourcing as a model for problem solving an introduction and cases. Convergence 14 (1), 75–90.
Breaux, T.D., Schaub, F., 2014. Scaling requirements extraction to the crowd: experiments with privacy policies. IEEE 22nd International Requirements Engineering Conference (RE).
Casamayor, A., Godoy, D., Campo, M., 2010. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. Inf. Softw. Technol. 52 (4), 436–445.
Chaos: Standish Group: Chaos Report, 2015.
Cheng, B.H., Atlee, J.M., 2007. Research directions in requirements engineering. In: May 2007 Future of Software Engineering, pp. 285–303.
Cleland-Huang, J., Dumitru, H., Duan, C., Castro-Herrera, C., 2009. Automated support for managing feature requests in open forums. Commun. ACM 52 (10), 68–74.
Cleland-Huang, J., Settimi, R., Zou, X., Solc, P., 2006. The detection and classification of non-functional requirements with application to early aspects. In: 14th IEEE International Requirements Engineering Conference (RE'06), pp. 39–48.
Cleland-Huang, J., Settimi, R., Zou, X., Solc, P., 2007. Automated classification of non–functional requirements. Requirements Eng. 12, 103–120.
Doan, A., Ramakrishnan, R., Halevy, A.Y., 2011. Crowdsourcing systems on the world-wide web. Commun. ACM 54 (4), 86–96.
Dzung, D.V., Ohnishi, A., 2009. Improvement of quality of software requirements with requirements ontology. In: 2009 Ninth International Conference on Quality Software, pp. 284–289.
Finkelstein, A., Lim, S.L., 2012. StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation. IEEE Trans. Softw. Eng. 38 (3), 707–735.
Galvis Carreño, L.V., Winbladh, K., 2013. Analysis of user comments: an approach for software requirements evolution. In: Proceedings of the 2013 International Conference on Software Engineering, May, pp. 582–591.
Glinz, M., 2007. On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 21–26.
Grady, R.B., Caswell, D.L.: Software metrics: establishing a company-wide program, 1987.
Haiduc, S., Arnaoudova, V., Marcus, A., Antoniol, G., 2016. The use of text retrieval and natural language processing in software engineering. In: Proceedings of the 38th International Conference on Software Engineering Companion, pp. 898–899.
Happel, H.J., Seedorf, S., 2006. Applications of ontologies in software engineering. In: Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC, pp. 5–9.
Hosseini, M., Phalp, K.T., Taylor, J., Ali, R., 2014. Towards crowdsourcing for requirements engineering, pp. 82–101.
ISO/IEC Standard 9126-1. Software Engineering – Product Quality – Part 1: Quality Model, 2001.
Jiao, F., Wang, S., Lee, C.H., 2006. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, pp. 209–216.
Keller, J.M., Gray, M.R., Givens, J.A., 1985. A fuzzy k-nearest neighbor algorithm. In: IEEE Trans. Syst. Man Cybernet., pp. 580–585.
Maalej, W., Nabil, H., August 2015. Bug report, feature request, or simply praise? on automatically classifying app reviews. In: 2015 IEEE 23rd international requirements engineering conference (RE), pp. 116–125.
McKinley, D., 2013. Functionality and Usability Requirements For a Crowdsourcing Task Interface That Supports Rich Data Collection and Volunteer Participation (Doctoral Dissertation. School of Information Management, Victoria University of Wellington.
Meth, H., Brhel, M., Maedche, A., 2013. The state of the art in automated requirements elicitation. Inf. Softw. Technol. 55 (10), 1695–1709.
Mylopoulos, J., Chung, L., Nixon, B., 1992. Representing and using nonfunctional requirements: A process-oriented approach. IEEE Trans. Softw. Eng. 18 (6), 483–497.
Naparat, D., Finnegan, P., 2013. Crowdsourcing software requirements and development: a mechanism-based exploration of 'opensourcing'. In: Proceedings of the Nineteenth Americas Conference on Information Systems. Chicago, Illinois, August 14-17.
Ormeño, Y.I., Panach, J.I., Condori-Fern, N., Pastor, Ó., 2013. Towards a proposal to capture usability requirements through guidelines. In: IEEE 7th International Conference on Research Challenges in Information Science (RCIS), pp. 1–12.
Pohl, K., Rupp, C., 2011. Requirements Engineering fundamentals: a Study Guide For the Certified Professional For Requirements Engineering Exam-Foundation level-IREB Compliant. Rocky Nook, Inc.
Rashwan, A., Ormandjieva, O., Witte, R., 2013. Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier. In: Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual, pp. 381–386.
Riaz, M., King, J., Slankas, J., Williams, L., 2014. Hidden in plain sight: automatically identifying security requirements from natural language artifacts. In: IEEE 22nd International Requirements Engineering Conference (RE), pp. 183–192.
Rish, I., 2001. An empirical study of the naive Bayes classifier. In: IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, Vol. 3, pp. 41–46.
Roman, G.C., 1985. A taxonomy of current issues in requirements engineering. Computer 18 (4), 14–23.
Royce, W.W., 1970. Managing the development of large software systems. In: proceedings of IEEE WESCON, 26, pp. 328–338.
SERC, 2017. (Systems Engineering Research Center): system Qualities ontology, tradespace and affordability (aqota) project-phase 5 Technical Report SERC-2017-TR-105, April 30.
Slankas, J., Williams, L., 2013. Automated extraction of non-functional requirements in available documentation. In: 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE), pp. 9–16.
Snijders, R., Atilla, Ö., Dalpiaz, F., Brinkkemper, S., 2015. Crowd-centric requirements engineering: a method based on crowdsourcing and gamification Technical Report Series, (UU-CS-2015-004).
Suykens, J.A., Vandewalle, J., 1999. Least squares support vector machine classifiers. Neural Process. Lett. 9 (3), 293–300.
Toro, A.D., Jiminez, B.B., Bonilla, M.T., Corchuelo, R., Cortés, A.R., Pérez, J., 1999. Expressing customer requirements using natural language requirements templates and patterns. In: *Proceedings of the 3rd IMACS/IEEE International Multiconference on Circuits, Systems,Communications, and Computers*. IEEE Computer Society.
Van Lamsweerde, A., 2009. Requirements engineering: from system goals to UML models to software specifications. Wiley Publishing.
Vlas, R.E., Robinson, W.N., 2012. Two rule-based natural language strategies for requirements discovery and classification in open source software development projects. J. Manag. Inf. Syst. 28 (4), 11–38.
V-ModellXT, 2004. Entwicklungsstandard Fur IT-Systeme Des Bundes. Bundesrepublic Deutschland, Corgehensmodell www.kbst.bund.de.
Wang, H., Wang, Y., Wang, J., 2014. A participant recruitment framework for crowdsourcing based software requirement acquisition. In: 2014 IEEE 9th International Conference on Global Software Engineering, pp. 65–73.
Zand, D.E., Sorensen, R.E., 1975. Theory of change and the effective use of management science. Admin. Sci. Q. 532–545.
Zhang, W., Yang, Y., Wang, Q., Shu, F., 2011. An empirical study on classification of non-functional requirements. In: The Twenty-Third International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), pp. 190–195.



Chuanyi Li is a full-time researcher at the Software Institute, Nanjing University. He received both his B.D. (2012) and Ph.D. (2017) from the Software Institute at Nanjing University. His research interests include software engineering, systems and software quality assurance, process modeling, simulation and improvement, process mining. His research results have been published in international journals and conference proceedings including JSS, Inf Sci, IEEE TSC, FGCS etc.

LiGuo Huang is an Associate Professor, Department of Computer Science and Engineering at the Southern Methodist University (SMU), Dallas, TX, USA. She received both her Ph.D. (2006) and M.S. from the Computer Science Department and Center for Systems and Software Engineering (CSSE) at the University of Southern California (USC). Her current research centers around process modeling, simulation and improvement, systems and software quality assurance and information dependability, mining systems and software engineering repository, stakeholder/value-based software engineering, and software metrics. Her research has been supported by NSF, U.S. Department of Defense (DoD) and NSA. She had been intensively involved in initiating the research on stakeholder/value-based integration of systems and software engineering. She can be contacted at lghuang@smu.edu.

Jidong Ge is an Associate Professor at Software Institute, Nanjing University. He received his PhD degree in Computer Science from Nanjing University in 2007. His current research interests include cloud computing, workflow scheduling, software engineering, workflow modeling, stochastic network optimization, process mining. His research results have been published in more than 50 papers in international journals and conference proceedings including IEEE TSC, JASE, FGCS, JSS, Inf Sci, ESA, SCIS, ICSE, SCC, APSEC, ICSSP, HPCC, SEKE, ICTAI etc.

Bin Luo is a Professor at the Software Institute, Nanjing University. His main research interests include cloud computing, computer network, workflow scheduling, and software engineering. His research results have been published in more than 50 papers in international journals and conference proceedings including IEEE TSC, ACM TIST, JSS, FGCS, Inf Sci, ESA, ICTAI etc. He is leading the institute of applied software engineering at Nanjing University.

Vincent Ng is an Associate Professor in the Department of Computer Science at the University of Texas at Dallas. He is also affiliated with the University's Human Language Technology Research Institute, where he conducts research on natural language processing and teaches undergraduate and graduate courses in machine learning. His current research interests include text mining and NLP applications, text clustering and categorization, opinion and argumentation mining, anaphora and co-reference resolution, etc.