The Journal of Systems & Software 178 (2021) 110965

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Machine learning based success prediction for crowdsourcing software projects☆

Inam Illahi [a], Hui Liu [a,*], Qasim Umer [a,c], Nan Niu [b]

[a] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
[b] Department of Electrical Engineering and Computer Science, University of Cincinnati, USA
[c] Department of Computer Science, COMSATS University Islamabad, Vehari, Pakistan

ARTICLE INFO

Article history:
Received 6 January 2020
Received in revised form 13 March 2021
Accepted 27 March 2021
Available online 20 April 2021

Keywords:
Competitive crowdsourcing
Classification
Machine learning
Risk
Prediction

ABSTRACT

Competitive Crowdsourcing Software Development is an online software development paradigm, promises the innovative, cost effective and high quality solutions on time. However, the paradigm is still in infancy and does not address the key challenges such as low rate of submissions and high risk of project failure. A significant number of software projects fail to receive a satisfactory solution and end up wasting the time and efforts of stakeholders. Therefore, the success prediction of a new software project may help stakeholders in the project crowdsourcing decision, saving their time and efforts. To this end, this study proposes a novel approach based on machine learning to predict the success of a software project for crowdsourcing platforms in terms of whether the given project will reach its completion or otherwise. First, the textual description and important attributes of software projects from TopCoder is extracted. Next, the description is preprocessed using natural language processing technologies. Then, keywords are identified using a modified keyword ranking algorithm and each software project is awarded a ranking score. Every software project is modeled as a vector that is based on the extracted attributes, its identified keywords and ranking scores. Using these vectors with their associated solution status, a support vector machine classifier is trained to predict the success of a given software project. Different machine learning classifiers are applied and it turns out that support vector machine yields the highest performance on the given dataset. Finally, the proposed approach is evaluated with history data of real software projects. The results of hold-out validation suggest that the average *precision, recall*, and *f-measure* are up to *94.53%, 99.30%* and *96.85%*, respectively.

1. Introduction

Competitive Crowdsource Software Development (CCSD) has gained tremendous attention in the software engineering community (Stol and Fitzgerald, 2014; Lakhani et al., 2006; Stol et al., 2018). It explores the possibility of replacing in-house software development to obtain cost-effective, innovative and high-quality solutions on time.

CCSD depends on an open call format (Mao et al., 2015), where clients (companies) crowdsource their software development projects (noted as projects for short in the rest of this paper) to CCSD platforms that arrange online competitions. The crowd (developers) participate in such competitions and present their innovative solutions (Fu et al., 2017) to win monetary rewards. There are numbers of CCSD platforms e.g., TopCoder, uTest, GetACoder, and Taskcn (Mao et al., 2015) that arrange online software development competitions. However, TopCoder[1] is the largest and widely trusted CCSD platform.

CCSD paradigm is still at an early stage and poses some challenges that remain to be resolved (Illahi et al., 2019). One of the biggest challenges it faces is that it compromises the success rate of a project. Resultantly, many projects are either not completed or fail to receive satisfactory solutions (Fitzgerald and Stol, 2015). For instance, TopCoder has 83% task quitting rate (projects that have registrants but do not have submitted solutions) from Jan 2014 to Jan 2015 (Yang et al., 2016). However, out of total submitted solutions, 81% projects are completed whereas 19% projects are failed due to different reasons, such as failed-screening and failed-review. One possible reason behind this challenge is the ecosystem of CCSD that mainly depends on the unknown, geographically distributed and uncontrolled crowds. Consequently, CCSD paradigm provides limited visibility and control over the progress of a project (Khanfor et al., 2017). Due to the limited access, the clients and CCSD platforms cannot predict whether

☆ Editor: Leandro Minku.
* Corresponding author.
E-mail address: liuhui08@bit.edu.cn (H. Liu).

[1] www.topcoder.com.

https://doi.org/10.1016/j.jss.2021.110965
0164-1212/

the project will be completed or not. Such occurrences cause dissatisfaction (Stol and Fitzgerald, 2014; Fitzgerald and Stol, 2015) among the clients and damage the popularity of the CCSD platforms. Therefore, an early and automatic success prediction of projects may help CCSD platform by avoiding the screening or reviewing the potentially rejected submissions. Different approaches have been identified to address this challenge. For instance, awarding the project to the right developer, exploring the developer's history before granting them a particular project and identifying the other influencing factors (Fu et al., 2017; Mao et al., 2015).

The detailed description of the projects has not yet been contemplated that are mostly described in natural language text and focuses some particular facets of a system, indicates some relevant concerns. The detailed description not only explains the functional requirements but also shows some concerns other than the functional requirements. Typical examples of these concerns are reliability, security, and performance, which may also be regarded as crosscutting concerns or early aspects (Rago et al., 2019). These concerns appear scattered in the detailed description. In traditional software development, analysts perform an early inspection to identify such keywords that denote any spurious concerns. For instance, keywords like "integration", "services" and "compatibility" refer to extensibility constraints for the system.

To this end, we deeply explore the detailed descriptions of the projects and propose a machine learning based novel approach for the success prediction of CCSD projects. The key insight is, extractive summarization methods can extract technical information (keywords and their ranking scores) from the textual description of projects. Notably, the proposed approach classifies the CCSD projects either as *success* or *failure*, where projects with completed and approved submissions are labeled as *success* otherwise *failure*. First, we extract projects from a CCSD platform *TopCoder* and preprocess their descriptions using natural language processing technologies. Second, we exploit and modify a keyword ranking algorithm to identify keywords and to compute their ranking scores. Third, given the extracted attributes of each project, its identified keywords, and their ranking scores, we present each project as a vector. Fourth, we train multiple machine learning classifiers for the success prediction of a given project. Finally, we evaluate each classifier with history data of crowdsourced projects. The results of hold-out validation suggest that the support vector machine based classifier outperforms other machine learning classifiers, and the average *precision*, *recall*, and *f-measure* are up to *94.53%*, *99.30%*, and *96.85%*, respectively.

The main contributions of the proposed approach are twofold:

- An automated machine learning based approach is proposed to the success prediction of a new project. To the best of our knowledge, it is the first approach to predict the successful completion of projects for crowdsourcing platforms.
- Evaluation results of the proposed approach suggest that the proposed approach is accurate, and the *precision*, *recall*, and *f-measure* are up to *94.53%*, *99.30%*, and *96.85%*, respectively.

The rest of the paper is organized as follows. Section 2 discusses the research background and 3 presents the proposed approach. Sections Section 4 describes the evaluation process and the results of the proposed approach. Related work is in Section 5. Threats to validity are discussed in Section 6. Finally, Section 7 concludes the paper along with future work direction.

2. Background

This section provides brief background knowledge of CCSD and explains the CCSD process.

2.1. Competitive Crowdsourced Software Development (CCSD)

CCSD is derived from *crowdsourcing*, a term jointly coined by Howe and Robinson in 2006 (Mao et al., 2017). CCSD is an emerging trend of outsourcing the development of software projects to the online community that is suitably skilled yet geographically distributed. CCSD facilitates various software development activities, starting from the requirement specification to the testing phase (Wu et al., 2013). It has gained significant popularity with many notable online portals like TopCoder, AppStori, and uTest, where thousands of software developers collaborate and compete to develop innovative software solutions (Stol et al., 2018; Hu and Wu, 2015). However, TopCoder has become the most popular CCSD platform having more than 1.2 billion global registered members (Stol et al., 2018; Hu and Wu, 2015).

2.2. CCSD process

The overview of the CCSD process is shown in Fig. 1. CCSD is an open call format where companies, who want to crowdsource their projects, request the crowdsourcing platforms to arrange online competitions to obtain cost-effective and innovative software solutions. Crowdsourcing platforms post the project's description and announce online competitions. Developers around the world get registered for the competitions that best match their expertise and compete to produce the best innovative software solutions to win the monetary rewards. After developers submit their solutions, the crowdsourcing platforms conduct a peer review to evaluate the quality of the solutions. Only the selected/satisfactory solutions are further evaluated to choose the best solution. Finally, the winners are rewarded with monetary rewards. Note that all projects that have zero submission or do not have any satisfactory solution are considered to be unsuccessful.

3. Approach

3.1. Overview

An overview of the proposed approach is presented in Fig. 2. The input of the proposed approach is a new CCSD project. The output is the successful prediction of a given CCSD project that is whether the given project will receive its solution or otherwise. The proposed approach works as follows:

- First, it collects CCSD projects from *TopCoder* and preprocesses the requirement document of each project using natural language processing technologies.
- Second, it exploits a text ranking algorithm with some modifications that extract the keywords and their ranking scores from each preprocessed requirement document.
- Third, it performs feature modeling using the extracted attributes of each project and its features (extracted keywords) to represent each project as a vector.
- Fourth, it trains a machine learning classifier for the success prediction of a new project.
- Finally, it predicts the success of a new project using the trained machine learning classifier.

The proposed approach is further detailed in the following sections.

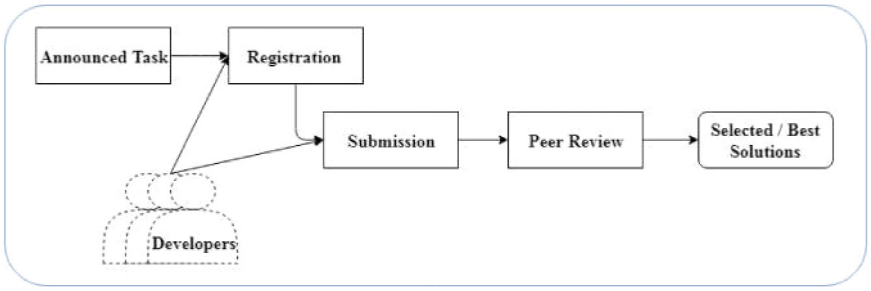

Fig. 1. Overview of CCSD project.

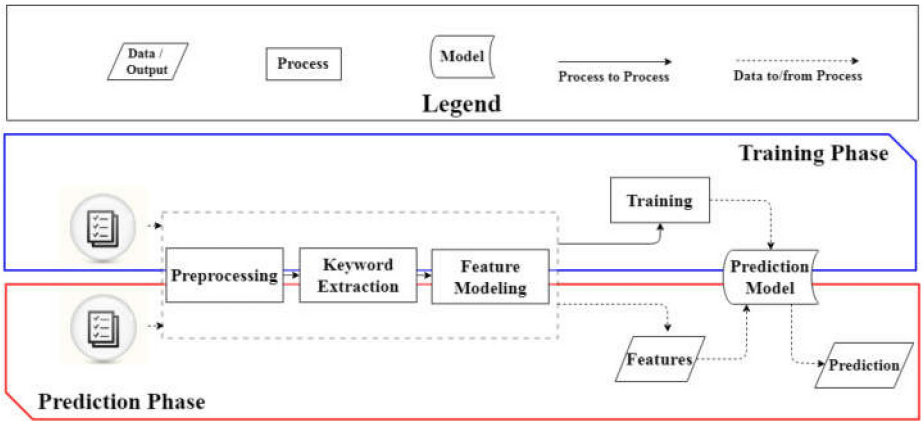

Fig. 2. Overview of the proposed approach.

3.2. Illustrating example

We use the following example to illustrate how the proposed approach performs success prediction. It is an example of TopCoder's software development project (30028416). It was started on *Jun* 6, 2013 and finished on *Jun* 15, 2013.

- Project name: "*Lycoming Website Dynamic Modules Build*".
- Start date: "*Jun 06, 2013*" is the date of the open call for the competition of the project.
- Submission date: "*Jun 15, 2013*" is the last date of submission for the software solution.
- Detailed requirements: (a snippet from requirements) "*The Lycoming is seeking a redesign of the existing website. The goal is to refresh the design, modernize the home page, and focus on usability. A key focus of the site should be the support and technical publications while maintaining focus on the main products — engines and parts. We have properly recreated the new Lycoming Engines website based on DotNetNuke Framework.*"
- Required technologies: "*.NET 4.0, ASP.NET, C#, SQL Server 2008*" are the technologies specified for developing the project.
- Required platforms: "*HTML*" is the required platform (could be more than one) on which the completed project would run.
- Status: "*not-completed*" is the success status of the project that indicates the project is successfully completed or not-completed.

In the following sections, we describe how the proposed approach works for the illustrating example.

3.3. Problem definition

A project k from a set of projects $\mathbb{K}$ can be formalized as in Eq. (1)

$$k = \langle d, t, p, l, r \rangle \tag{1}$$

where d, t, and p are the numbers of required days, technologies, and platforms to develop k, l is the length of requirement document, and r is the detailed requirement document of k.

For the illustrating example presented in Section 3.2, we have

$$k_e = \langle d_e, t_e, p_e, l_e, r_e \rangle \tag{2}$$

where,

- $d_e = 9$ is the number of required days. It is computed by subtracting the start date from the submission date.
- $t_e = 4$ is the number of required technologies.
- $p_e = 1$ is the number of required platforms.
- $l_e = 237$ is the total number of lines appeared in r_e.
- r_e = "*The Lycoming is seeking a redesign of the existing website. The goal is to refresh the design, modernize the home page, and focus on usability. A key focus of the site should be the support and technical publications, while maintaining focus on the main products-engines and parts. We have properly recreated the new Lycoming Engines website based on DotNetNuke Framework. The goal of this competition is to clearly identify all issues for the new Lycoming Website.*" It is the snippet of requirements of the project.

The proposed approach classifies given project k into *success* or *failure*. The class *success* represents that k is completed; i.e., one or more satisfying solutions are received. The class *failure* represents that k fails to receive its satisfactory solution. The success prediction could be represented as a function f:

$$c = f(k) \tag{3}$$

$$c \in \{success, failure\}, \quad k \in \mathbb{K} \tag{4}$$

where c is the classification result (e.g., *success* or *failure*) and f is a categorizing function.

3.4. Preprocessing

Natural language processing technologies are applied for the preprocessing of requirement documents extracted from TopCoder. The *html* tags are removed using a Python package *Beautifulsoup*. The preprocessing techniques include tokenization, stop-word removal, parts-of-speech (POS) tagging, negation handling, spelling correction, modifier word recognition, word inflection, and lemmatization. The following preprocessing layers are employed on the requirement document of each project.

- Tokenization: is the process of splitting text into tokens (words). We remove special characters (e.g., *punctuation marks*), decompose the text into words, and convert them into lowercase.
- Spell Correction: textual documents may have spelling mistakes. Therefore, spell correction is performed to remove all mistakes in documents.
- Stop-word removal: This layer removes those natural language words that have either very little meaning or not related to the structure such as *and*, *the*, *a*, *an*, and *are*.
- POS Tagging: A POS tag is assigned to each of the tokenized words from requirement documents.
- Word Inflection and Lemmatization: Word inflection converts the words into their singular form. For instance, the word *issues* is converted into *issue*. Lemmatization converts the words, mainly the nouns and adjectives, into their base words. For example; lemmatization converts the word *glasses* into *glass*.

To perform preprocessing, python natural language toolkit (NLTK) is used. After preprocessing, a project k can be represented as

$$k' = \langle d, t, p, l, ws \rangle \tag{5}$$

$$ws = \langle w_1, w_2, w_3, \ldots.., w_n \rangle \tag{6}$$

where ws are the words (tokens) from the requirement document of k after preprocessing. For the illustrating example presented in Section 3.2, we have

$$k'_e = \langle 9, 237, 4, 1, lycoming, seek, \ldots, website \rangle \tag{7}$$

where *lycoming*, *seek*, ….., *website* are the preprocessed words from k_e.

3.5. Keywords extraction

The submission of the software solutions and their evaluation is subject to the implementation of the required features written in requirements documents. Recent studies (Mihalcea and Tarau, 2004; Goldberg, 2016) suggest that machine learning techniques can extract and rank keywords according to the semantic representation of the text. Various techniques are available for extractive summarization and ranking keywords e.g., *TextRank* (Mihalcea and Tarau, 2004), *RAKE* (Rose et al.), *IBM NLU* (Vergara et al., 2017), *KEA* (Witten et al., 1999), and *YAKE* (Campos et al., 2018). However, *YAKE* yields the most accurate results in automatic keyword extraction for a single document (Campos et al., 2018). As the text used in the software projects is domain-dependent and *YAKE* is domain-independent. Therefore, some modifications are made in *YAKE* model to devise a new mechanism for identifying and ranking the keywords. The following steps are taken to capture the characteristics of each word w_i from Eq. (6).

3.5.1. Casing

The modified version of *YAKE* always considers the nouns and verbs if they are starting from a capital letter or acronyms. Therefore, we select the words starting with a capital letter, acronyms, nouns, or verbs for casing. Another reason for their selection is that most of them are important and required features in the requirement documents. The weight of the casing step is calculated as follows,

$$CW = \frac{max(TF(U(w_i)), TF(A(w_i), TF(N(w_i)), TF(V(w_i))))}{log_2(TF(w_i))} \tag{8}$$

where CW is the casing weight of w_i, $TF(U(w_i))$ is the frequency of w_i starting with an uppercase letter, $TF(A(w_i))$ is the frequency of w_i defined as an acronym, $TF(N(w_i))$ is the frequency of w_i defined as a noun, $TF(V(w_i))$ is the frequency of w_i defined as a verb, and $TF(w_i)$ is the frequency of w_i.

3.5.2. Positioning

Unlike the *YAKE*, all the words are considered equal and no extra weight is assigned to w_i based on its position. The dependent features of the requirement document are not considered in the modified *YAKE*.

The positioning weight is calculated as in Eq. (9)

$$PW = \frac{\sum Sen_{w_i}}{T{-}Sen} \tag{9}$$

where PW is the positioning weight of w_i, Sen_{w_i} is the position of the set of sentences in which w_i occurs, and T-*Sen* is the total number of sentences in which w_i occurs.

3.5.3. Frequency

In this step, the frequency of each w_i is computed under the assumption that higher the frequency, more important is w_i. This step similar to that in *YAKE* and the frequency weight is computed as given below

$$FW = \frac{TF(w_i)}{MeanTF + 1 * \sigma} \tag{10}$$

where FW is the frequency of w_i and *MeanTF* is the means of w_n frequencies, and σ is their standard deviation.

3.5.4. Relatedness

Unlike the *YAKE*, it is not necessary to compute the relatedness of w_i to the context as the stop-words have been removed in the preprocessing step of the proposed approach. Therefore, there is no point to quantify the extent to which a w_i resembles the properties of a stop-word.

3.5.5. Frequency in different sentence

In this step, the frequency of w_i appearing within different sentences is calculated

$$FDW = \frac{SF(w_i)}{\#Sentences} \tag{11}$$

where FDW is the frequency weight of w_i in different sentences and $SF(w_i)$ is the sentence frequency of the word w_i.

3.6. Ranking of extracted keywords

Based on the four characteristics (*relatedness* is ignored as clarified in Section 3.5.4) of w_i, the rank R of w_i can be assigned as in Eq. (12)

$$R_{w_i} = \frac{PW}{CW + FW + FDW} \tag{12}$$

For the illustrating example presented in Section 3.2, *YAKE* returns keywords *website*, *seek*, …., *dotnetnuke* with their ranking scores 0.0147, 0.0211, …., 0.1194.

3.7. Feature modeling

Finally, a high dimension matrix is created wherein each project is a row of the matrix. The columns of the matrix are the number of required days d, technologies t, platforms p, length of the requirement document l, and extracted keywords of a complete set of project. A feature vector of a project k can be defined as in Eq. (13)

$$k = \langle d, t, p, l, f_1, f_2, f_3, \ldots.., f_n \rangle \tag{13}$$

where, f_1, f_2, f_3, and f_n are the ranking scores of the extracted keywords. For the example presented in 3.2, we have

$$k_e = \langle 9, 237, 4, 1, website, seek, \ldots., dotnetnuke \rangle \tag{14}$$

where *website*, *seek*, …., *dotnetnuke* are the extracted keywords.

For each project, we use a ranking score to represent a feature in the feature vector. If a keyword is not found in the project, we mark it 0. Otherwise, we assign its ranking score to a keyword. For the illustrating example presented in Section 3.2, we have the following feature vector.

$$k'_e = \langle 9, 237, 4, 1, 0.0147, 0, \ldots.. \rangle \tag{15}$$

where 0.0147, 0, ….. are assigned values based on the existence of keyword (features).

3.8. Training and prediction

The proposed approach utilizes the support vector machine (SVM) to capture the relationship between the features and the success status of projects. SVM can handle the high dimensional feature space and removes the irrelevant features (Umer et al., 2018). The SVM classification depends on the principle of Structural Risk Minimization (SRM) principle that comes from computational learning theory (Vapnik, 2000), where SRM finds a hypothesis to guarantee the lowest true error. To seek a decision surface, SVM requires a binary training set and separates both classes into n-dimensional space called hyperplane (Umer et al., 2018). As SVM has a significant impact on the classification of the documents (Scholkopf et al., 1999; Vapnik, 1999), we train it using the feature set of each project that is tagged during feature modeling using Eq. (4). We test the trained model to predict the success status of a new project.

Training

To construct a classification model that assigns an unlabeled project to a class c as defined in the Eq. (4), a set of projects $\mathbb{K}$ is given. Each project contains its classification category c and set of features as mentioned in Eq. (13). We train SVM and use success/failure projects to seek the decision surface for the prediction. The decision surface is a hyperplane that is used for training of the proposed approach. Given the project k_i labeled into specified binary categories y_i, find a weight vector $\mathbf{w}$ such that discriminant function separates the categories for $i = 0, 1$ and can be formalized as in Eq. (16)

$$f(\mathbf{k_i}) = \mathbf{w}^\top \mathbf{k_i} + b \tag{16}$$

where f is a discriminant function that computes the decision surface for the training data k_i, $\mathbf{w}$ is a weight vector that represents the normal to decision surface and b is bias.

Prediction

Once the vector is defined with the training set, each project from the testing dataset may be succeeded (*success*) if

$$\mathbf{w}^\top \mathbf{k} + b > 1 \tag{17}$$

or *failure* otherwise.

4. Evaluation

In this section, we evaluate the proposed approach on the crowdsourcing projects collected from *TopCoder*.

4.1. Research questions

The evaluation investigates the following research questions:

- RQ1: How accurate is the proposed approach in success prediction of projects?
- RQ2: Does re-sampling influence the performance of the proposed approach? If yes, to what extent?
- RQ3: Does *SVM* outperform other machine learning algorithms in predicting the success of projects?
- RQ4: Does preprocessing influence the performance of the proposed approach? If *yes*, to what extent?
- RQ5: What is the most influential input to the proposed approach?
- RQ6: Does the contest's prize significantly influence the success' change of the contest?

The first research question (RQ1) computes the accuracy of the proposed approach. We compare the proposed approach against the two baseline prediction algorithms: *random prediction algorithm* and *zero rule algorithm*. Both algorithms are the most commonly used ones when the researchers do not have any baseline approach for comparison while working on very uncommon or rare problem. Random prediction algorithm requires the distinct actual outcome values from the training data and predicts the random outcome values for the testing data. Whereas, zero rule algorithm predicts the most frequently occurring classification in a set of data. To answer the second research question (RQ2), we investigate the effect of re-sampling as our dataset is skewed to favor accepted successful projects (4 times more accepted projects than rejected projects). There are three different ways to perform re-sampling: under-sampling, over-sampling, or by setting the different value of classifier threshold. Data is added and reduced in an imbalanced dataset for over-sampling and under-sampling, respectively. Similarly, the dataset could be balanced using the threshold value of classifier that assigns weight of each class. Note that we balance our dataset using both over-sampling and under-sampling to investigate the effect of re-sampling.

The third research question (RQ3) compares the performance of different machine learning and deep learning algorithms. The comparison may reveal whether SVM outperforms other machine learning and deep learning algorithms in predicting the success of projects.

To answer the fourth research question (RQ4), we investigate the impact of the preprocessing on the given inputs. We provide two inputs: preprocessed text of requirement documents and keywords (extracted by applying the state of the art model explained in Section 4.3) to the machine learning model for the success prediction.

To answer the fifth research question (RQ5), first, we compute the performance of the proposed approach by eliminating the textual and non-textual features from the feature vector. Then, we compare their results to figure out the most influential input factor in the proposed approach.

To answer the sixth research question (RQ6), first, we compute the performance of the proposed approach by including/excluding the contest prize as feature. Then, we compare the result to check the influence of contest's prize on the proposed approach.

Table 1
Description of dataset.

Total number of projects	16190
Success projects	13113 (81%)
Failure projects	3077 (19%)
Duration of projects	Up to July 2018

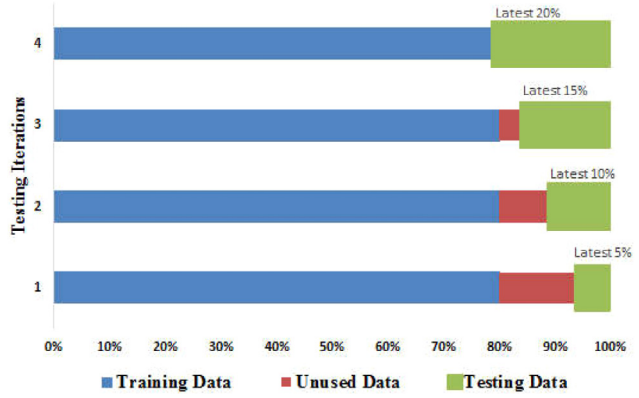

Fig. 3. Evaluation process.

4.2. Dataset

The history data of software development projects is extracted from *TopCoder*. We use *TopCoder Rest API*.[2] to extract the required attributes, such as project starting date, ending date, required technologies and platforms, requirements, and status of the publicly available projects till July 2018 and save into the database. The overview of the dataset is presented in Table 1 The *status* attribute of a project returned from TopCoder API specifies whether a project is completed or not. Since the *status* can have different values for instance *completed*, *canceled-winner unresponsive*, *canceled-failed screening*, *canceled-failed review*, we reduce this multi-class classification problem to a binary classification problem. To this end, we classify (label) the completed-projects having accepted-submissions as *positive*, whereas the completed-projects having rejected-submissions are classified as *negative*. The total numbers of projects are 16190, out of which 81% are successful and 19% are *failed*. The projects whose submission is *incomplete* or status attribute is not defined or the requirements attribute is empty or referring to another web link are not included.

4.3. Process

The evaluation of the proposed approach is performed as follows. First, we extract the projects $\mathbb{K}$ from TopCoder, sort them by date, and apply natural language processing technologies for preprocessing mentioned in Section 3.4. Next, we perform the hold-out validation technique on $\mathbb{K}$ and split $\mathbb{K}$ into 80%–20%. Where 80% of all sorted projects are taken as training set K_{train}. 20% of recent projects are taken as testing set K_{test}. Notably, the reason behind the selection of hold-out validation instead of k-fold cross-validation is that we would like to leverage the history data to new projects. If k-fold validation was employed, the prediction of a given project (noted *testProj*) may leverage the project histories that are not yet available when *testProj* is proposed on the platform. However, we cannot leverage histories of projects that do not yet exist when the to-be-predicted project comes in. To this end, we first sort the extracted *TopCoder* data in the ascending order year-wise. Then, we train the machine learning classifier on first 80% data (the elder data) and test it on the other data (more recent data) as shown in Fig. 3. As a result, while making prediction on a given project *testProj*, we only leverage the project histories that was available when *testProj* was proposed on the crowdsourcing platform. We split the testing projects into i^{th} combinations notated as $m_i (i = 1...4)$ and m_1, m_2, m_3, and m_4 contain recent 5%, 10%, 15%, and 20% projects out of the total testing projects. Finally, we validate the trained classifier with each i^{th} combination of the testing set.

For each m_i, a step by step process is as follows:

- First, we select the training set K_{train}.
- Second, a Multinomial Naive Bayes classifier (MNB) is trained on K_{train}.
- Third, a Linear Regression classifier (LR) is trained on K_{train}.
- Fourth, a Random Forest Classifier (RF) is trained on K_{train}.
- Fifth, a Support Vector Machine (SVM) classifier is trained on K_{train}.
- Sixth, a Convolutional Neural Network (CNN) classifier is trained on K_{train}.
- Seventh, a Long Short Term Memory (LSTM) classifier is trained on K_{train}.
- Eighth, for each i^{th} combination of testing set K_{test}, we predict the success status of each project from K_{itest} using trained classifiers (NB, MNB, LR, RF, SVM, CNN, and LSTM), respectively.
- Finally, we calculate the precision, recall, and f-measure for each classifier to compare their performances.

4.4. Metrics

To evaluate the approach, we apply most widely used metrics *precision*, *recall*, and *f-measure* for binary classification:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$f-measure = \frac{2 * precision * recall}{precision + recall}$$

where *precision*, *recall*, and *f-measure* are respectively, precision, recall, and f-measure of the approach for success prediction of the projects $\mathbb{K}$. *TP* is the number of the projects that are truly predicted as success, *TN* is the number of projects that are correctly predicted as fail, *FP* is the number of the projects that are incorrectly predicted as success, and *FN* is the number of projects that are incorrectly predicted as fail.

4.5. RQ1: Accuracy of the proposed approach

To answer the research question **RQ1**, we compare the *proposed approach* with two baseline algorithms (*random prediction* and *zero rule*). Both algorithms are used alternatively as a benchmark to check the accuracy of the proposed approach. As the proposed approach does not have any existing approach for the performance comparison, it is the first approach for predicting the success of projects to the best of our knowledge. Therefore, we choose these algorithms for the performance comparison of the proposed approach.

[2] https://tcapi.docs.apiary.io.

Table 2
Comparison against alternative approaches.

	Proposed approach			Random prediction			Zero rule		
Testing data	Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
Latest 5%	96.15%	98.63%	97.38%	51.49%	51.65%	51.57%	84.92%	100.00%	91.84%
Latest 10%	92.76%	99.33%	95.93%	85.89%	87.03%	86.46%	82.58%	100.00%	90.46%
Latest 15%	94.27%	99.56%	96.84%	51.29%	51.32%	51.31%	95.93%	100.00%	97.92%
Latest 20%	94.94%	99.68%	97.25%	64.23%	64.57%	64.40%	91.84%	100.00%	95.75%
Average	**94.53%**	**99.30%**	**96.85%**	**63.23%**	**63.64%**	**63.43%**	**88.82%**	**100.00%**	**93.99%**

4.5.1. Results

Evaluation results of *proposed approach*, *random prediction*, and *zero rule* are presented in Table 2. The first column presents the testing iterations. We evaluate each of the classifiers on ith iteration (where $i = 1, 2, 3, 4$) using *5%*, *10%*, *15%*, and *20%* recent projects from the dataset. Columns 2–10 present the performance results of *precision*, *recall*, and *f-measure* for each classifier, respectively. Rows present the performance of a particular classifier on each iteration, respectively. The last row presents the average results.

The average precision, recall, and f-measure of the *proposed approach*, *random prediction*, and *zero rule* are (94.53%, 99.30%, and 96.85%), (63.23%, 63.64%, and 63.43%), and (88.82%, 100.00%, and 93.99%), respectively. Note that we have not found *overfitting* problem during the evaluation of the proposed approach as the variation between the training and testing accuracies is minor.

4.5.2. Observations

From Table 4, we make the following observations:

- The *proposed approach* outperforms the *random prediction* and *zero rule* classifiers.
- In precision, the performance improvement of the *proposed approach* upon *random prediction* and *zero rule* is *49.50% = (94.53%–63.23%) / 63.23%* and *16.96% = (94.53%–80.82%) / 80.82%*, respectively.
- In recall, the performance improvement of the *proposed approach* upon *random prediction* and *zero rule* is *56.03% = (99.30%–63.64%) / 63.64%* and *% = (−0.07%–100.00%) / 100.00%*, respectively. The reason of performance decrease of the *proposed approach* in recall against *zero rule* is that *zero rule* always predicts the majority class (class of successful project in our case).
- In f-measure, the performance improvement of the *proposed approach* upon *random prediction* and *zero rule* is *52.69% = (96.85%–63.43%) / 63.43%* and *3.04% = (96.85%–93.99%) / 93.99%*, respectively.

4.5.3. Impact of sorting

We perform an additional experiment to compare the evaluation techniques (cross-validation and hold-out). To this end, we evaluate the proposed approach on both evaluation techniques with and without sorting. The average *precision*, *recall*, and *f-measure* of hold-out validation with and without sorting are (*94.53%*, *99.30%*, and *96.85%*) and (*92.95%*, *97.82%*, and *95.32%*), respectively. The performance improvement of the proposed approach with sorted dataset in *precision*, *recall*, and *f-measure* is *1.70% = (94.53%–92.95%) / 92.95%*, *1.51% = (99.30%–97.82%) / 97.82%*, and *1.61% = (96.85%–95.32%) / 95.32%*, respectively.

The results suggest that the proposed approach works better on hold-out validation with sorting.

The reason behind that is the selection hold-out validation to leverage the history data to new projects as discussed in Section 4.3.

Consequently, both validation approaches perform better with sorting of the projects, however, hold-out validation technique is significant among them. The reason of this significance is that most of the iteration of 10-fold cross-validation train the machine with features that are related to new technology and do not have impact while testing old projects. Notably, based on this experiment, we select the hold-out validation for the remaining analysis.

Table 3
Impact of re-sampling.

Re-sample	Precision	Recall	F-Measure
No	94.53%	99.30%	96.85%
Under-sampling	96.55%	99.42%	97.96%
Over-sampling	95.18%	99.37%	97.23%

4.5.4. Misclassification

We also observed that the proposed approach shows many false positive and false negative results. For example, failed projects (*30064333* and *30062477*) are predicted as *successful* projects. Whereas, the successful projects (*30064319* and *30056740*) are predicted as *failed* projects. A comprehensive keyword extraction tool specific to software engineering text may avoid such misclassification and improve the performance of the proposed approach. However, we have not yet fully understood the rationale of the misclassification. In future, we shall investigate the rationale for misclassification, and figure out the measures to reduce the misclassification.

Based on the preceding analysis, we conclude that the proposed approach is accurate in success prediction of projects.

RQ2: Impact of re-sampling

Re-sampling adjusts the class distribution and corrects the bias within an imbalanced dataset. To investigate the impact of re-sampling, we use both over-sampling and under-sampling. We adopt the synthetic minority over-sampling approach for the over-sampling of the insignificant class, whereas we randomly select n-samples from the significant class for under-sampling. Notably, we use an equal number of accepted and rejected projects in order to avoid bias. Moreover, we sort the dataset by date to create real testing condition.

Evaluation results of with and without re-sampling are presented in Table 3. The first column presents the re-sampling settings. Columns 2–4 present the performance results of *precision*, *recall*, and *f-measure* on different re-sampling settings, respectively. Each row presents the average performance against each re-sampling setting. The average *precision*, *recall*, and *f-measure* the proposed approach with under-sampling and over-sampling are (*96.55%*, *99.42%*, and *97.96%*) and (*95.18%*, *99.37%*, and *97.23%*), respectively. Table 3 shows the difference in performance between balanced and imbalanced datasets. It suggests that both techniques of re-sampling does improve the overall performance of the proposed approach.

For further performance evaluation of the proposed approach, we also plot a receiver operation characteristics (ROC) curve (shown in Fig. 4) on different under-sampling rates of majority

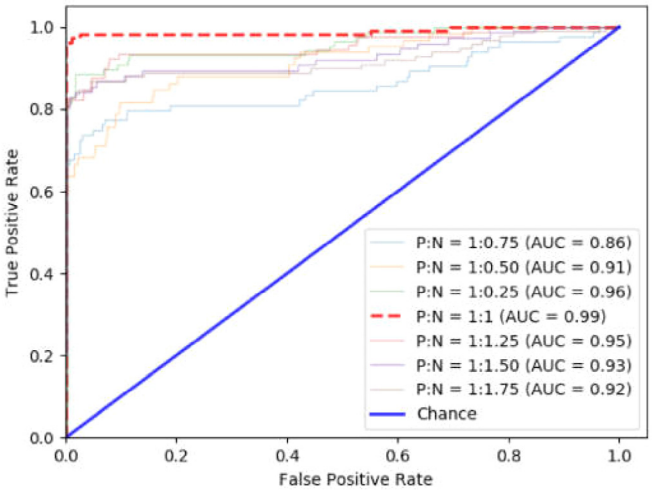

Fig. 4. ROC on different re-sampling rates.

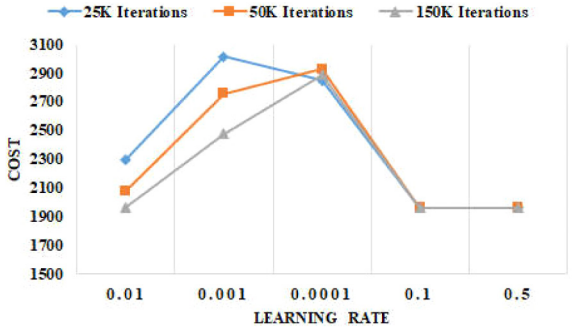

Fig. 5. RF convergence against learning rates.

class and over-sampling rates of the minority class. The ROC curve illustrates the trade-off between true positive rate (sensitivity) and specificity (1 - false positive rate). The performance curve of an approach that is closer to the top-left corner indicates its significance. The average value of the ROC curve (AUC = 0.99) against the ratio 1:1 of accepted/rejected projects suggests that the proposed approach has the best measure of separability with equal distribution of input projects.

4.6. RQ3: Performance comparison of machine learning algorithms

To answer the research question **RQ3**, we apply widely used text classification algorithms (*MNB*, *LR*, *RF*, *SVM*, *CNN*, and *LSTM*) due to their competitive performance (Wu et al., 2008; Sohrawardi et al., 2014; Ramay et al., 2019). We also exploit the mentioned *CNN* and *LSTM* models as these are reported effective for text classification (Ramay et al., 2019).

4.6.1. Parameter tuning of classification algorithms

The tuning the parameters of machine learning algorithms is usually used in the context of statistical model fitting. To tune the above mentioned classification algorithms, we train and evaluate them with different internal parameter settings to identify the best parameter settings. Furthermore, we compare the performances of the above classification algorithms with the best parameter settings. We select the *MNB* among *Multinomial*, *Bernoulli*, and *Gaussian* Naive Bayes because it considers a feature vector where a given term represents the number of times it appears or very often i.e. frequency. Notably, *MNB* is not only best in performance (as shown in Table 4) but also suitable for feature modeling of the proposed approach (as mentioned in Section 3.7). In contrast, *Bernoulli* Naive Bayes is a binary algorithm used when the feature is present or not and *Gaussian* Naive Bayes is based on continuous distribution.

Moreover, we train and evaluate *LR* on different learning rates (*0.5*, *0.1*, *0.01*, *0.001*, *0.0001*) and number of iterations (*25000*, *50000*, and *150000*), respectively. From Fig. 5, it can be observed that with the given data, learning rate (*0.5*) converges after *25000* iterations, whereas learning rate (*0.0001*) takes more time and cost for the convergence after *150000* iterations. Similarly, we set the *max_depth* = *3*, *min_sample_split* = *2*, and *max_terminal_nodes* = *25* for *RF*. It is observed that the increase in *max_depth* and *min_sample_split* decreases the performance of the proposed approach. However, the tree is underfitting when the value of *max_terminal_nodes* is very small, whereas the performance of the tree increases as the value of *max_terminal_nodes* increases. Notably, the tree starts to overfit as the value of *max_terminal_nodes* goes beyond *25*. Furthermore, the proposed approach (*SVM*) is tested with different *kernel* parameter settings: *kernel* = *linear* for linear classification and *kernel* = (*rbf*, *poly*, or *sigmoid*) for the non-linear classification. We select the *kernel* = *linear* because of its best performance on the proposed approach.

For the deep learning approaches, we use three layers of *CNN* with *filter size* = 128, *kernel size* = 1, and *activation* = *tanh*. Similarly, our *LSTM* model contains embedding layer, LSTM layer (*dropout* = 0.2 and *recurrent dropout* = 0.2), and dense layer (*activation* = *sigmoid*). We use the *binary crossentropy* as a loss function. Notably, although the change in the performances of proposed approach is minor across different parameter values of the classification algorithms, we select the best parameter values collected from the parameter tuning process for the comparison of the classification algorithms.

4.6.2. Results

Evaluation results of *SVM*, *LR*, *RF*, *MNB*, *CNN*, and *LSTM* are presented in Table 4. The first column presents the classifiers. Columns 2–4 present the performance results of *precision*, *recall*, and *f-measure* of each classifier, respectively. Each row presents the average performance of a particular classifier. The average precision, recall, and f-measure of the *SVM*, *LR*, *RF*, *MNB*, *CNN*, and *LSTM* are (94.53%, 99.30%, and 96.85%), (93.96%, 98.27%, and 96.06%), (93.86%, 97.49%, and 95.64%), (91.75%, 84.46%, and 87.94%), (82.64%, 86.16%, and 84.36%), and (78.34%, 85.52%, and 81.69%), respectively. The results of applying these classifiers suggest that *SVM* yields most accurate results on our dataset.

We also present the *f-measure* distribution of validation results for *SVM*, *LR*, *RF*, *MNB*, *CNN*, and *LSTM* using beanplot (Fig. 6). A beanplot combines the boxplot, denstity plot, and rug plot. It shows the density curve which is more informative than a boxplot. We compare the *f-measure* distributions by plotting one bean for each classifier that represents its density. Across a bean, each short horizontal line represents the *f-measure* on a single classifier, whereas the long horizontal line represents the average *f-measure*. Notably, we implement the beanplot in *R* using *beanplot* package. From Fig. 6, we observe that the average *f-measure* (long horizontal line) of the proposed classifier *SVM* is better than the highest *f-measure* (short horizontal lines) of all other classifiers.

4.6.3. Observations

From Table 4, we make the following observations:

- *SVM* outperforms the *LR*, *RF*, *MNB*, *CNN*, and *LSTM* in *precision*, *recall*, and *f-measure*, respectively. Notably, boosting algorithms are not exploited to correct the classification errors due to required additional computational cost, sensitive to noisy data, e.g., AdaBoost, and less susceptible to the overfitting problem than most learning algorithms.

Table 4
Comparison against Different Machine Learning Algorithms.

	Precision	Recall	F-measure
SVM	94.53%	99.30%	96.85%
LR	93.96%	98.27%	96.06%
RF	93.86%	97.49%	95.64%
MNB	91.75%	84.46%	87.94%
CNN	82.64%	86.16%	84.36%
LSTM	78.34%	85.52%	81.69%

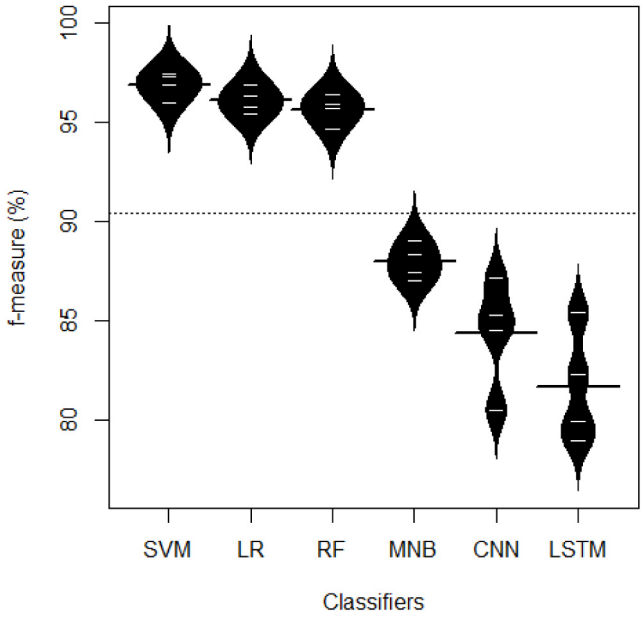

Fig. 6. Distribution of accuracy.

- The *SVM* achieves best results for different reasons. First, *SVM* creates a hyperplane in feature space (Li et al., 2009; Khan et al., 2010). The hyperplane has maximum margin for all projects (outliers are exceptions). For text classification, it helps *SVM* in better generalization of the testing data in contrast to distance-based and similarity-based algorithms such as *RF* (Cristianini and Shawe-Taylor, 2000). Second, linear *SVM* helps it to search different combinations within features and performs classification with low computational complexity in contrast to other kernel options of *SVM* (Li et al., 2009). Moreover, *SVM* works better than other machine learning classifiers for long text classification, e.g., *LR*, *RF*, and *MNB*) (Khan et al., 2010).
- *LR* outperforms *MNB* and its performance is very close to the proposed classifier *SVM*. The possible reason behind this difference is large features set of projects of *LR*. *LR* is a discriminative model that performs better with large training feature size in contract to *MNB* which is a generative model (Ng and Jordan, 2002). It is reported that *LR* works same as *SVM* when the kernel of *SVM* is *Linear* (Li et al., 2009). Therefore, *LR* may outperform the proposed classifier *SVM* with large dataset.
- *LR* also outperforms *RF*. The possible reason behind this is *LR*'s fast training attribute and its execution on the sparse features (a feature having almost zero values represents a sparse feature). We observe that the difference between the performances of *LR* and *RF* is not significant. However, the curse of dimensionality (Ng and Jordan, 2002; Khan et al., 2010) in *LR* may increase its performance significantly on a larger dataset. Because the degree of freedom in *RF* makes it inappropriate for high dimensional features in success prediction of projects.
- Although deep learning algorithms, such as *CNN* and *LSTM*, achieve better results in comparison to machine learning algorithms for different classification problems (Ramay et al., 2019), however, they do not outperform the proposed machine learning classifier (*SVM*). A possible reason is that we have a small size dataset for the evaluation of the proposed approach and deep learning classifiers perform better with large training datasets (Ramay et al., 2019). Another reason is the unsupervised nature of the deep learning classifiers. It is important to find the key functionalities/keywords from the textual documents of projects to predict their success. However, the deep learning classifiers perform classification on the whole text by finding a relationship and it decreases the performance of deep learning classifiers. Therefore, we propose the keywords as features (mentioned in Section 3.5), which is gist of our proposed approach.

Moreover, we perform Wilcoxon test in Stata software with its built-in settings to calculate the difference between approaches and analyze these differences. The results suggest p-value < (alpha = 0.05) is true for *f-measure*, where p-values for *SVM* against *LR*, *RF*, *MNB*, *CNN*, and *LSTM* are *0.038*, *0.037*, *0.029*, *0.025*, and *0.024*, respectively.

Furthermore, we compute the effect size to investigate the difference between approaches by employing Cohen's delta d, where d >= 0.2, d >= 0.5, and d >= 0.8 indicates the difference as small, medium, and large, respectively. Result suggests that the difference of the proposed approach is *small* in contrast to *LR* and *RF*, whereas the difference of the proposed approach is *medium* with the remaining approaches. Notably, we apply the Wilcoxon test and effect size *pairwise* as both tests are used for comparison of two distributions. To this end, we select *SVM* and one from other algorithms alternatively as two populations to apply Wilcoxon test and effect size.

Based on the preceding analysis, we conclude that the results of the proposed approach are significant with the SVM classifier.

4.7. RQ4: Influence of preprocessing

The requirement document of each project contains some irrelevant and meaningless data e.g., punctuation and stop-words (as mentioned in Section 3.4). Passing such data to machine learning algorithms is an overhead. It reduces their performance and increases the computational cost. To answer the research question **RQ4**, we compare the results of the proposed approach by enabling and disabling preprocessing.

4.7.1. Results

The evaluation results are presented in Table 5. The first column of the table presents the preprocessing input settings. Columns 2–4 present the performance results of *precision*, *recall*, and *f-measure*. Rows present the performance of the proposed approach with different preprocessing input settings. The last row presents the performance improvements of the proposed approach with different preprocessing input settings.

The average precision, recall, and f-measure of the *proposed approach* by enabling preprocessing are (94.53%, 99.30%, and 96.85%) and disabling preprocessing are (93.64%, 74.90%, and 83.23%).

Table 5
Influence of Preprocessing.

Preprocessing	Precision	Recall	F-measure
Enabled	94.53%	99.30%	96.85
Disabled	93.64%	74.90%	83.23
Improvement	**0.95%**	**32.58%**	**16.36%**

Table 6
Comparison of Different Input Factors.

	Precision	Recall	F-measure
Default	94.53%	99.30%	96.86%
Disabling Non-textual Features	92.50%	98.13%	95.23%
Disabling Textual Features	86.21%	89.87%	88.00%

4.7.2. Observations

From Table 5, we make the following observations:

- The proposed approach with preprocessing layers attains significant improvement in performance. The evaluation results suggest that the performance improvement in *precision*, *recall*, and *f-measure* is *0.95%*, *32.58%*, and *16.36%*, respectively.
- Disabling preprocessing significantly decreases the performance in *recall* of the proposed approach from *99.30%* to *74.90%*. One possible reason for the decrease in performance is that applying the proposed approach without preprocessing may include unwanted words (stop-words or unstemmed words) as features, affecting the efficiency and processing time of the proposed approach.

Based on the preceding analysis, we conclude that the deep preprocessing layers of the textual information (requirements) of the projects is an essential and vital step for the proposed approach.

4.8. RQ5: Influence of different input factors

To answer the research question **RQ5**, we compare the results of the proposed approach by enabling and disabling different inputs.

4.8.1. Results

The evaluation results are presented in Table 6. The first column of the table presents the input settings. Where, columns 2–4 present the performance results of *precision*, *recall*, and *f-measure*. Rows present the performance of the proposed approach upon different input settings, respectively. The average precision, recall, and f-measure of the *proposed approach* with different settings (*default*, *disabling non-textual features*, and *disabling textual features*) are (94.53%, 99.30%, and 96.85%), (92.50%, 98.13%, and 95.23%), and (86.21%, 89.87%, and 88.00%), respectively.

4.8.2. Observations

From Table 6, we make the following observations:

- Textual features alone (i.e., without non-textual features) is often insufficient for success prediction. Compare to default settings (i.e., enabling both textual and non-textual features), disabling non-textual features decrease *precision* significantly from *94.53%* to *86.21%*, *recall* from *99.30%* to *89.87%*, and *f-measure* from *96.86%* to *88.00%*.
- Second, disabling textual features (i.e., non-textual features alone) also results in a significant reduction in performance of the proposed approach. It significantly decreases *precision* from *94.53%* to *92.50%*, *recall* from *99.30%* to *98.13%*, and *f-measure* from *96.86%* to *95.23%*.

From the preceding analysis, we conclude that both textual and non-textual features are critical for the success prediction of projects, and disabling one out of both will reduce the performance significantly of the proposed approach.

To investigate why the textual features are so useful in success prediction, we manually analyze the textual features. The in-depth manual analysis suggests that some keywords may influence the success/failure of the projects. To illustrate the observation, consider the following example (snippet) from success projects:

> *"This application will integrate with the existing FMS system to get the achievement and payment data from the system batch validation against a set of pre-defined rules (that is external). The GUI of this application will provide simple, but useful screens that will allow filtering and viewing/validating of the batch validated achievements and payments. After the employees and validation attributes are selected and displayed, the authorized users can review and validate and log validation actions to the web application. The users do no update the achievements and payments on the web application, the users only log the actions to be taken on the web application, the adjustment of each and payments are done outside the tool".*

In the example above, we find the candidate keyword **validate**. This word often suggests that the specification of the project has explicitly specified the requirements as well as the test criteria. We find that most of (91%) the projects whose textual specification contain keyword **validate** succeed finally. Notably, lemmatization converts the words *validation*, *validating*, and *validated* into *validate* as mentioned in Section 3.4.

The following example (snippet) is a typical failure project:

> *"Simple e-mail component to be used by visual basic 6 is a simple component to be used by applications written in visual basic 6 to send unlimited numbers of e-mails along with attached files limited by a configurable maximum size, the component functions provide the simple mail transfer protocol (SMTP) to send the e-mails, also should provide the SSL as a security protocol. The component will receive in run time (each time a new mail is created): Mail server information. Mail message information (including attachments). The component writes all the successful and unsuccessful processes to a log. The component is compatible with Windows XP, Vista, 7 in both platforms 32 and 64 bit"*

Within this specification, we find two keywords: **SSL** and **compatible**. Such keywords specify the security and compatibility criterion. Such criterion, however, requires significant efforts to accomplish, especially with a stringent time constraint. As a result, most of the developers withdraw from such projects or fail to accomplish such projects.

Based on the manual analysis of the textual features, we observed that inclusion of the security and compatibility related keywords decreases the success rate of projects and restricts novice developers to participate because security and compatibility features are bit hard to explain and require significant efforts and time to accomplish. The existing literature also shows (Fitzgerald and Stol, 2015; Stol et al., 2018) that in CCSD services like Topcoder, a significant amount of time of stackholders is wasted in explaining the requirements to the competitive software developers. Moreover, most of the senior developers do not make the essential contributions especially for the complex task (Illahi et al., 2019). The preemptive measures against such occurrences may attract more senior software developers and use of simple and plain words for explaining requirement may increase the success rate of projects.

Table 7
Influence of Contest's Prize.

Contest's prize	Accuracy	Precision	Recall	F-measure
Included	95.08%	94.84%	99.56%	97.25%
Excluded	94.49%	94.53%	99.30%	96.85%

Furthermore, we investigate the significance of each keyword having discriminating effect on predicting success. The list of the keywords that are most associated with the accepted projects and the rejected projects are (validate (91%), configuration (92%), sequence (95%), edit (94%), and documentation (92%)) and (ultimate (90%), secure (91%), realtime (90%), compatible (92%), and protect (89%)), respectively. We leverage *TF/IDF* to measure the significance of keywords in a project and sort the significant keywords with the highest occurrences in the projects. We combine all the successful projects to compute the term frequency (TF) and take the complete set of projects to compute the inverse document frequency (IDF). The likelihood of a keyword in success prediction (reject prediction) can be formalized as,

$$L(k_i, sp) = \frac{count(d_{sp}, k_i)}{count(d_{total}, k_i)} \tag{18}$$

where, $L(k_i, sp)$ is the likelihood of keyword $K - i$ is successful project sp, $count(d_{ns}, k_i)$ is the number of successful projects that includes keyword k_i, and $count(d_{total}, k_i)$ is the total number of projects that includes keyword k_i.

4.9. RQ6: Influence of contest's prize

To investigate the reasons of misclassifications, we further compare the results of the proposed approach by introducing the contest's prize as feature. The reason of selecting contest's prize for the investigation of misclassification is that it can distort the classification, as it increases the success' chances (as it attracts more developers and motivates them) when it is high and the other way around.

4.9.1. Results and observations

The evaluation results are presented in Table 7. From this table, we make the following observations:

- First, introducing contest's prize slightly reduces misclassification. Comparing against the default setting, including contest's prize slightly increases *accuracy* from 94.49% to 95.08%. We also notice that the same is true for other performance metrics, i.e., precision, recall, and F-measure.
- Second, introducing contest's prize could not significantly reduce misclassification. One possible reason is that contest's prize is time dependent, e.g., old projects have lower prize in contrast to new projects. Another possible reason is that as suggested by existing survey (Illahi et al., 2019), participants on TopCoder often pay scant attention to the prize. Consequently, increasing prize has little impact on the success of projects.

From the preceding analysis, we conclude that introducing contest's prize has minor influence on the performance of the proposed approach.

5. Threats

5.1. Threats to validity

The metrics which are chosen to evaluate our approach can be a threat to construct validity. We have used the precision, recall, and f-measure for the evaluation of the proposed approach. However, these metrics are the most widely used and adopted by many researchers (Tian et al., 2015).

Another threat to validity is related to the parameter values of the classification algorithms. We did experiments to find the optimal parameter settings instead of using default ones. Notably, we select the best parameter settings for the comparison of the classification algorithms as specified in Section 4.6.1. However, the results may decrease with the change in selected parameter settings.

A constructive validity threat is the adoption of YAKE for keywords extraction and assigning their corresponding scores to the keywords. There are few other tools available but we select YAKE as it is the latest model and its results outperform other existing models. Although we modify *YAKE* to mitigate this threat. However, comprehensive keywords extracting tool for software engineering text may decrease the performance of the approach.

An internal validity threat is related to the implementation of the approach. To mitigate the threat of implementation, results are cross-checked. However, there could be some unnoticed errors.

A threat to external validity is related to the generalizability of our approach. We have only considered and analyzed the software development projects from TopCoder platform. Projects from other crowdsourcing platforms may increase or decrease the performance of the approach.

An external validity threat to the approach is, it may not perform well or may not work for the projects written in other languages. The proposed approach is trained and evaluated on projects written in English.

Another threat to external validity is a small number of projects. Therefore, we use traditional machine learning algorithms to evaluate the proposed approach. Deep learning algorithms allow many parameters to be adjusted and mostly require significant training data.

6. Related work

CCSD has significantly gained the attention of the research community in recent years. Researchers have explored CCSD from its model to domain application (Mao et al., 2015). However, despite the importance of resolution prediction of project completion, there are only a few studies that focus on project completion issue. However, some state-of-the-art studies are discussed below.

Prediction of project similarity: a cluster-based calcification and competitive network boosting approach (CBC-CN) is introduced by Fu et al. (2017) that routes a project to a right developer. Their approach selects the most similar projects, builts a classifier based on these similar projects, and recommends a list of candidates. They employ a cluster-based classification method to handle the local characteristics in CCSD projects. The technique considers similar projects together based on the similarity of content. From historical activities, they construct the competitive network of developers and re-rank the initial order.

Prediction of project developers: a machine learning based approach (PREM) introduced by Mao et al. (2016) that actively matches the best suitable developer to the available project on TopCoder. They identify two typical types of developers in CCSD and observe their behavior to construct a prestige network. They employ a multi-class and single-label classification techniques (Mao et al., 2015). Similarly, they employ content-based recommendation techniques and develop a framework named CrowdRex. The framework automatically recommends developers for newly arriving CCSD projects. They employ multi-class, single-label classification technique where a developer is treated as a class. Based on the historical activities of the developers, the

machine learner captures the characteristics of each developer and recommends the best-matched developers to newly added projects.

Prediction of influential factors: Khanfor et al. (2017) used 4,872 TopCoder's projects from Jan 2014 to Jan 2015 and applied traditional machine learning techniques to identify the most influential factors of the project outcomes. The framework detects cancelation-prone projects that facilitate management to prevent potential delay. Based on three different types of analysis, i.e., market analysis (number of projects created), project completion analysis (percentage of completed projects), and attractiveness analysis(average number of crowd applying for a project) (Dubey et al., 2016), they identify factors that influence project completion on TopCoder and Upwork. They also identify factors that influence the quality of the completed projects. Moreover, a decision support system (DCS-DS) introduced by (Yang et al., 2016), investigates the influencing factors of crowd software developer's behaviors in a CCSD context. They introduced dynamic features extracted to characterize dynamic competition factors and proposed an analytics-based dynamic worker decision support framework. Saremi et al. (2017) proposed a set of questions to improve team elasticity in adaptive software development on TopCoder that impacts workers' performance with different skill and experience.

Moreover, not limited to CCSD, some studies are conducted on micro crowdsourcing platforms. For instance, based on matrix factorization, (Yuen et al., 2012) also proposed a task recommendation framework for task preference modeling and preference-based task recommendation for Amazon MTurk, which is a micro-task crowdsourcing platform. Similarly based on Matrix factorization, (Jung, 2014) proposed methods to rout a crowdsourcing task to a most appropriate worker to maximize accuracy on the new task of Amazon MTurk. Moreover, to prone out the dishonest worker on Amazon Turk, (Ye et al., 2015) proposed two classifications based on task type and task reward amount. They proposed a trust evaluation model that consists of two types, i.e., task type based trust (TaTrust) and reward amount based trust (RaTrust). Finally, they introduced an evolutionary algorithm MOWS-GA based on NSGA-II, which effectively differentiate honest workers and dishonest workers. Although, the studies we have discussed in this section are focused on CCSD. However, these studies are different from our proposed approach.

7. Conclusion and future work

CCSD paradigm is widely employed that promises innovative, cost-effective and high-quality solutions on time. However, it faces some key challenges such as low rate of solution submissions. Consequently, a significant number of crowdsourcing projects did not get any satisfying solution and ended up with the wasting of time and efforts of the developers, CCSD platform and companies who crowdsourcing their software projects. To this end, we propose a machine learning based approach for success prediction of a CCSD project that whether it may get a satisfactory solution or not. We perform preprocessing using natural language processing technologies, employ the modified keyword ranking algorithm to identify the keywords and their ranking scores for each CCSD project, model the each CCSD project as a vector, and train a support vector machine classifier that predicts whether a given CCSD project will receive its solution or not. This will help companies in saving their time and efforts. The proposed approach is evaluated with history data of CCSD projects. The results of hold-out validation suggest that the average *precision*, *recall*, and *f-measure* are up to *94.53%*, *99.30%*, and *96.85%*, respectively.

The impact of our work is to show that the requirement description of projects significantly helps in their success prediction. However, it would be interesting to find out the impact of project description in the success prediction of project if reworded. Furthermore, we would also like to explore other metadata features of projects to improve the performance of the proposed approach using deep learning approaches.

CRediT authorship contribution statement

Inam Illahi: Methodology, Data curation, Formal analysis, Writing - original draft. **Hui Liu:** Conceptualization, Methodology, Paper revision, Response to reviewers. **Qasim Umer:** Algorithm implementation and evaluation. **Nan Niu:** Conceptualization, Paper revision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (61690205, 61772071).

References

Campos, R., Mangaravite, V., Pasquali, A., Jorge, A.M., Nunes, C., Jatowt, A., 2018. Yake! collection-independent automatic keyword extractor. In: Pasi, G., Piwowarski, B., Azzopardi, L., Hanbury, A. (Eds.), Advances in Information Retrieval. Springer International Publishing, Cham, pp. 806–810.

Cristianini, N., Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, http://dx.doi.org/10.1017/CBO9780511801389.

Dubey, A., Abhinav, K., Taneja, S., Virdi, G., Dwarakanath, A., Kass, A., Kuriakose, M., 2016. Dynamics of Software Development Crowdsourcing. In: 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE), pp. 49–58. http://dx.doi.org/10.1109/ICGSE.2016.13.

Fitzgerald, B., Stol, K.-J., 2015. The Dos and Don'ts of Crowdsourcing Software Development, Vol. 8939. http://dx.doi.org/10.1007/978-3-662-46078-8_6.

Fu, Y., Sun, H., Ye, L., 2017. Competition-aware task routing for contest based crowdsourced software development. In: 2017 6th International Workshop on Software Mining (SoftwareMining). pp. 32–39. http://dx.doi.org/10.1109/SOFTWAREMINING.2017.8100851.

Goldberg, Y., 2016. A primer on neural network models for natural language processing. J. Artif. Int. Res. 57 (1), 345–420, http://dl.acm.org/citation.cfm?id=3176748.3176757.

Hu, Z., Wu, W., 2015. Game theoretic analysis for offense-defense challenges of algorithm contests on topcoder. In: 2015 IEEE Symposium on Service-Oriented System Engineering. pp. 339–346. http://dx.doi.org/10.1109/SOSE.2015.44.

Illahi, I., Liu, H., Umer, Q., Zaidi, S.A.H., 2019. An empirical study on competitive crowdsource software development: Motivating and inhibiting factors. IEEE Access 7, 62042–62057. http://dx.doi.org/10.1109/ACCESS.2019.2915604.

Jung, H.J., 2014. Quality assurance in crowdsourcing via matrix factorization based task routing. In: Proceedings of the 23rd International Conference on World Wide Web. In: WWW '14 Companion, ACM, New York, NY, USA, pp. 3–8. http://dx.doi.org/10.1145/2567948.2567951, http://doi.acm.org/10.1145/2567948.2567951.

Khan, A., Baharudin, B., Lee, L.H., Khan, K., Tronoh, U.T.P., 2010. A review of machine learning algorithms for text-documents classification. J. Adv. Inf. Technol..

Khanfor, A., Yang, Y., Vesonder, G., Ruhe, G., Messinger, D., 2017. Failure prediction in crowdsourced software development. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC). pp. 495–504. http://dx.doi.org/10.1109/APSEC.2017.56.

Lakhani, K.R., Lohse, P.A., Panetta, J.A., Karim, C., Lakhani, R., Lohse, P.A., Panetta, J.A., Lakhani, K.R., Lohse, P.A., Panetta, J.A., 2006. The value of openness in scientific problem solving.

Li, Y., Bontcheva, K., Cunningham, H., 2009. Adapting SVM for data sparseness and imbalance: a case study in information extraction. Nat. Lang. Eng. 15 (2), 241–271. http://dx.doi.org/10.1017/S1351324908004968.

Mao, K., Capra, L., Harman, M., Jia, Y., 2017. A survey of the use of crowdsourcing in software engineering. J. Syst. Softw. 126, 57–84. http://dx.doi.org/10.1016/j.jss.2016.09.015, http://www.sciencedirect.com/science/article/pii/S0164121216301832.

Mao, K., Wang, Q., Jia, Y., Harman, M., 2016. RN / 16 / 06 PREM : Prestige network enhanced developer-task matching for crowdsourced software development august 9 , 2016.
Mao, K., Yang, Y., Wang, Q., Jia, Y., Harman, M., 2015. Developer recommendation for crowdsourced software development tasks. In: 2015 IEEE Symposium on Service-Oriented System Engineering. pp. 347–356. http://dx.doi.org/10.1109/SOSE.2015.46.
Mihalcea, R., Tarau, P., 2004. Textrank: Bringing order into text. In: Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing. http://aclweb.org/anthology/W04-3252.
Ng, A.Y., Jordan, M.I., 2002. On discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (Eds.), Advances in Neural Information Processing Systems 14. MIT Press, pp. 841–848, http://papers.nips.cc/paper/2020-on-discriminative-vs-generative-classifiers-a-comparison-of-logistic-regression-and-naive-bayes.pdf.
Rago, A., Diaz-Pace, J.A., Marcos, C., 2019. Do concern mining tools really help requirements analysts? An empirical study of the vetting process. J. Syst. Softw. 156, 181–203. http://dx.doi.org/10.1016/j.jss.2019.06.073, http://www.sciencedirect.com/science/article/pii/S0164121219301359.
Ramay, W., Umer, Q., Yin, X.C., Zhu, C., Illahi, I., 2019. Deep neural network-based severity prediction of bug reports. IEEE Access 7, 46846–46857. http://dx.doi.org/10.1109/ACCESS.2019.2909746.
Rose, S.J., Engel, D.W., Cramer, N.O., Cowley, W.E., Automatic Keyword Extraction from Individual Documents.
Saremi, R., Yang, Y., Ruhe, G., Messinger, D., 2017. Leveraging crowdsourcing for team elasticity: an empirical evaluation at topcoder. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). pp. 103–112. http://dx.doi.org/10.1109/ICSE-SEIP.2017.2.
Scholkopf, B., Burges, C.J.C., Smola, A.J. (Eds.), 1999. Advances in kernel methods: Support vector learning. MIT Press, Cambridge, MA, USA.
Sohrawardi, S., Azam, I., Hosain, S., 2014. A comparative study of text classification algorithms on user submitted bug reports. In: Ninth International Conference on Digital Information Management (ICDIM 2014). pp. 242–247. http://dx.doi.org/10.1109/ICDIM.2014.6991434.
Stol, K., Caglayan, B., Fitzgerald, B., 2018. Competition-based crowdsourcing software development: A multi-method study from a customer perspective. IEEE Trans. Softw. Eng. http://dx.doi.org/10.1109/TSE.2017.2774297, 1–1.
Stol, K.-J., Fitzgerald, B., 2014. Two's company, three's a crowd: A case study of crowdsourcing software development. In: Proceedings of the 36th International Conference on Software Engineering. In: ICSE 2014, ACM, New York, NY, USA, pp. 187–198. http://dx.doi.org/10.1145/2568225.2568249, http://doi.acm.org/10.1145/2568225.2568249.
Stol, K.-J., Fitzgerald, B., 2014. Two's company, three's a crowd: A case study of crowdsourcing software development. In: Proceedings of the 36th International Conference on Software Engineering. In: ICSE 2014, ACM, New York, NY, USA, pp. 187–198. http://dx.doi.org/10.1145/2568225.2568249, http://doi.acm.org/10.1145/2568225.2568249.
Tian, Y., Lo, D., Xia, X., Sun, C., 2015. Automated prediction of bug report priority using multi-factor analysis. Empir. Softw. Eng. 20 (5), 1354–1383. http://dx.doi.org/10.1007/s10664-014-9331-y, http://dx.doi.org/10.1007/s10664-014-9331-y.
Umer, Q., Liu, H., Sultan, Y., 2018. Emotion based automated priority prediction for bug reports. IEEE Access 6, 35743–35752. http://dx.doi.org/10.1109/ACCESS.2018.2850910.
Vapnik, V.N., 1999. The Nature of Statistical Learning Theory.
Vapnik, V., 2000. The nature of statistical learning theory. Stat. Eng. Inf. Sci. 8, 1–15. http://dx.doi.org/10.1007/978-1-4757-3264-1_1.
Vergara, S., El-Khouly, M., Tantawi, M.E., Marla, S., Sri, L., 2017. Building cognitive applications with IBM watson services: Volume 7 natural language understanding. In: Building Cognitive Applications with IBM Watson Services: Volume 7 Natural Language Understanding. https://www.redbooks.ibm.com/redbooks/pdfs/sg248398.pdf.
Witten, I.H., Paynter, G.W., Frank, E., Gutwin, C., Nevill-Manning, C.G., 1999. KEA: Practical automatic keyphrase extraction. In: Proceedings of the Fourth ACM Conference on Digital Libraries. In: DL '99, ACM, New York, NY, USA, pp. 254–255. http://dx.doi.org/10.1145/313238.313437, http://doi.acm.org/10.1145/313238.313437.
Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J., Steinberg, D., 2008. Top 10 algorithms in data mining. Knowl. Inf. Syst. 14 (1), 1–37. http://dx.doi.org/10.1007/s10115-007-0114-2, https://doi.org/10.1007/s10115-007-0114-2.
Wu, W., Tsai, W.-T., Li, W., 2013. An evaluation framework for software crowdsourcing. Front. Comput. Sci. 7 (5), 694–709. http://dx.doi.org/10.1007/s11704-013-2320-2, http://dx.doi.org/10.1007/s11704-013-2320-2.
Yang, Y., Karim, M.R., Saremi, R., Ruhe, G., 2016. Who should take this task?: Dynamic decision support for crowd workers. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. In: ESEM '16, ACM, New York, NY, USA, pp. 8:1–8:10. http://dx.doi.org/10.1145/2961111.2962594, http://doi.acm.org/10.1145/2961111.2962594.
Ye, B., Wang, Y., Liu, L., 2015. Crowd trust: A context-aware trust model for worker selection in crowdsourcing environments. In: 2015 IEEE International Conference on Web Services. pp. 121–128. http://dx.doi.org/10.1109/ICWS.2015.26.
Yuen, M.-C., King, I., Leung, K.-S., 2012. Task recommendation in crowdsourcing systems. In: Proceedings of the First International Workshop on Crowdsourcing and Data Mining. In: CrowdKDD '12, ACM, New York, NY, USA, pp. 22–26. http://dx.doi.org/10.1145/2442657.2442661, http://doi.acm.org/10.1145/2442657.2442661.

Inam Illahi graduated from University of Sargodha, Pakistan 2007. He completed his MS degree in 2010 from Chalmers University of Technology, Sweden. He is pursuing his PhD from the School of Computer Science and Technology, Beijing Institute of Technology, China. He is particularly interested in software maintenance, crowdsourcing and machine learning.

Hui Liu is a professor at the School of Computer Science and Technology, Beijing Institute of Technology, China. He received BS degree in control science from Shandong University in 2001, MS degree in computer science from Shanghai University in 2004, and PhD degree in computer science from the Peking University in 2008. He was a visiting research fellow in centre for research on evolution, search and testing (CREST) at University College London, UK. He served on the program committees and organizing committees of prestigious conferences, such as ICSME, RE, ICSR, and COMPSAC. He is serving as associate editor for IEEE Access and IET Software, and guest editor for Empirical Software Engineering and Journal of Systems and Software. He is particularly interested in software refactoring, AI-based software engineering, and software quality. He is also interested in developing practical tools to assist software engineers.

Qasim Umer received the BS degree in computer science from Punjab University, Pakistan in 2006, MS degree in .net distributed system development from University of Hull, UK in 2009, and MS degree in computer science from University of Hull, UK in 2012. He is currently perusing PhD degree in computer science from Beijing Institute of Technology, China. He is particularly interested in machine learning, data mining and software maintenance.

Nan Niu is an Associate Professor in the Department of Electrical Engineering and Computer Science, University of Cincinnati, USA. His research interests include software requirements engineering, scientific software development, and human-centric computing. He received the Ph.D. degree in Computer Science from the University of Toronto in 2009. He is a member of ACM and ASEE, and a Senior Member of IEEE. Contract him at nan.niu@uc.edu.