11-2018

Recommending who to follow in the software engineering Twitter space

Abhabhisheksh SHARMA
Singapore Management University, abhisheksh.2014@smu.edu.sg

Yuan TIAN
Singapore Management University, ytian@smu.edu.sg

Agus SULISTYA
Singapore Management University, aguss.2014@smu.edu.sg

Dinusha WIJEDASA
Singapore Management University, dwijedasa@smu.edu.sg

David LO
Singapore Management University, davidlo@smu.edu.sg

Published in ACM Transactions on Software Engineering and Methodology,
Vol 27, Issue 4, November 2018, Pages 16–33
DOI 10.1145/3266426

Recommending Who to Follow in the Software Engineering Twitter Space

ABHISHEK SHARMA, YUAN TIAN, AGUS SULISTYA, DINUSHA WIJEDASA, and DAVID LO, School of Information Systems, Singapore Management University, Singapore

With the advent of social media, developers are increasingly using it in their software development activities. Twitter is one of the popular social mediums used by developers. A recent study by Singer et al. found that software developers use Twitter to "keep up with the fast-paced development landscape." Unfortunately, due to the general-purpose nature of Twitter, it's challenging for developers to use Twitter for their development activities. Our survey with 36 developers who use Twitter in their development activities highlights that developers are interested in following specialized software gurus who share relevant technical tweets.

To help developers perform this task, in this work we propose a recommendation system to identify specialized software gurus. Our approach first extracts different kinds of features that characterize a Twitter user and then employs a two-stage classification approach to generate a discriminative model, which can differentiate specialized software gurus in a particular domain from other Twitter users that generate domain-related tweets (aka domain-related Twitter users). We have investigated the effectiveness of our approach in finding specialized software gurus for four different domains (JavaScript, Android, Python, and Linux) on a dataset of 86,824 Twitter users who generate 5,517,878 tweets over 1 month. Our approach can differentiate specialized software experts from other domain-related Twitter users with an F-Measure of up to 0.820. Compared with existing Twitter domain expert recommendation approaches, our proposed approach can outperform their F-Measure by at least 7.63%.

CCS Concepts: • **Software and its engineering → Collaboration in software development**;

Additional Key Words and Phrases: Twitter, software engineering, recommendation systems

ACM Reference format:
Abhishek Sharma, Yuan Tian, Agus Sulistya, Dinusha Wijedasa, and David Lo. 2018. Recommending Who to Follow in the Software Engineering Twitter Space. *ACM Trans. Softw. Eng. Methodol.* 27, 4, Article 16 (October 2018), 33 pages.
https://doi.org/10.1145/3266426

1 INTRODUCTION

Twitter is a popular social media platform and is continuously gaining traction and users. As of July 2017, Twitter has a total of more than 328 million active monthly users who generate about 500 million short messages (aka tweets or microblogs) daily [54]. Twitter allows users to post short

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.
Authors' address: A. Sharma, Y. Tian, A. Sulistya, D. Wijedasa, and D. Lo, School of Information Systems, Singapore Management University, 81 Victoria St., Singapore, VA, 188065, Singapore; emails: {abhisheksh.2014, ytian, aguss.2014, dwijedasa, davidlo}@smu.edu.sg.

1049-331X/2018/10-ART16 $15.00
https://doi.org/10.1145/3266426

messages that are broadcasted to other users who have chosen to *follow* them. These messages can be further *retweeted* (i.e., propagated) to reach an even larger number of Twitter users. Additionally, users can *mention* other users (by specifying user names prefixed by the "@" symbols) or attach *hashtags* (keywords prefixed by the "#" symbols) in their tweets. Twitter allows users to get fast up-to-date information about recent events and is a powerful platform for information sharing, having characteristics at the intersection of news media and social networks [22].

Twitter and general social media channels have revolutionized the way developers work and interact with one another. Singer et al. surveyed 271 GitHub developers and found that Twitter "helps them keep up with the fast-paced development landscape" [45]. Among their respondents, more than 70% of them used Twitter to help them stay current about the latest technologies, practices, and tools they use, and learn things that they aren't actively looking for. Furthermore, a majority of the respondents used Twitter to connect to and build trust with other developers, and a significant percentage of respondents used Twitter to build communities around software development projects. The survey highlighted the increasing role that Twitter plays in the professional activities of software developers.

Despite the benefit brought by Twitter, its enormous size poses a number of challenges for its users, including software developers. Singer et al. highlighted that a central challenge faced by developers is to maintain a relevant network. Due to the fact that following other users is the preferred way of getting information from Twitter (besides search), not carefully curating the network might make it hard for developers to find relevant information that is interesting and useful. To validate this challenge, Singer et al. surveyed developers for their experience in using Twitter. Seventy-two percent of the respondents in their survey agree that they carefully consider whom they would want to follow. Unfortunately, finding suitable users to follow among the more than 328 million users in Twitter is not an easy feat.

In this work, we would like to help developers find interesting people to follow. To accomplish this goal, we first surveyed about 38 developers to better understand developers' needs. For 36 of them who actively use Twitter in their development activities, we asked them about the kinds of Twitter accounts they would like to follow (see Section 3). Our survey questionnaire was open ended and developers were free to enter any type of account that they wanted to follow. We find that more than 75% of the 36 respondents prefer to follow specialized software gurus in their domains of interest. Our finding is in line with that of Singer et al., who observed that many developers follow thought leaders from their technological niches [45].

To follow up on this finding, we propose an automated approach that can identify specialized software gurus from a large number of Twitter users. Our criteria for a specialized software guru is that he/she must be an experienced software developer in a specialized domain, and he or she must have shared useful information for other developers in the specialized domain. We include the last criterion since a guru (meaning teacher in Sanskrit) must have imparted knowledge to others. Also, it would be pointless to follow an expert developer who never shares something useful.

Our guru recommendation system identifies software gurus by first finding a subset of Twitter users who are potentially interested in software development and who generate *domain-related tweets* (i.e., tweets mentioning a particular domain of interest, e.g., Python). Our approach then extracts different kinds of features from each user in this set of *domain-related users* (i.e., users that generate domain-related tweets). These features can be grouped into four families: Content, Network, Profile, and GitHub. Based on these features, this candidate user set is then further analyzed by a two-stage classification process, which generates a discriminative model (aka a classifier) that differentiates specialized software gurus from other domain-related users.

To evaluate the main contribution of this work, which is a new approach that identifies specialized software gurus on Twitter, we have considered four domains of interest (JavaScript, Android,

Python, and Linux) and analyzed a collection of 5,517,878 tweets. These tweets were generated by 86,824 Twitter users and were collected over a 1-month period. The evaluation results show that our approach can differentiate between specialized software gurus and other domain-related users with an F-Measure score of 0.820 (for JavaScript gurus), 0.681 (for Android gurus), 0.602 (for Python gurus), and 0.522 (for Linux gurus), respectively. Our approach outperforms the state-of-the-art domain-specific Twitter expert recommendation approaches by Pal and Counts [33], as well as Klout [38], and achieves higher scores on metrics of precision, recall, and F-Measure. The improvement in F-Measure scores is by at least 7.63% (for Linux gurus). The effectiveness of our approach has been evaluated based on the following research questions, which are discussed in detail in Section 6:

- **RQ1:** How effective is our specialized software guru recommendation approach?
- **RQ2:** Can our approach outperform existing Twitter expert recommendation approaches?
- **RQ3:** What are important features that better differentiate specialized software gurus from nongurus?
- **RQ4:** Which feature values have the best predictive power across each domain?
- **RQ5:** What is the cross-domain performance of our approach?

The structure of the remainder of the article is as follows. We discuss related work in Section 2. In Section 3, we present the design and results of our online survey to find out who developers would like to follow on Twitter. In Section 4, we introduce the features that we use to characterize domain-related users on Twitter. In Section 5, we present our proposed approach that finds specialized software gurus on Twitter. We describe the settings and results of the experiments that evaluate our proposed approach in Section 6. We discuss the results in Section 7. We conclude and mention future work in Section 8.

2 RELATED WORK

Singer et al. surveyed 271 and interviewed 27 active developers on GitHub [45]. They found that many developers are using Twitter to "keep up with the fast-paced development landscape." Specifically, developers used Twitter to get awareness of people and trends, extend their technical knowledge, and build connections with other developers. Their paper also presents two major challenges affecting software developers using Twitter (i.e., maintaining a relevant network and consuming content), which highlight the need to help developers to find suitable users to follow from the massive number of Twitter users. Our study is motivated by this need.

Researchers have performed a number of empirical studies on microblogs posted in Twitter (aka tweets) [5, 51, 52, 60]. Bougie et al. analyzed 11,679 tweets posted by 68 developers from three open-source projects [5]. They observed that software engineers leverage Twitter to communicate and share information. Wang et al. analyzed 568 tweets posted by developers from the Drupal open-source project [60]. They found that Drupal developers use Twitter to coordinate efforts, share knowledge, encourage potential contributors to join, and so forth. Tian et al. manually categorized 300 tweets that contained software-related hashtags into 10 groups, which include commercial, news, tools and code, question and answer, events, personal, opinion, tips, job, and miscellaneous [51]. Prasetyo et al. manually analyzed a sample of 300 tweets and labeled each of them as a software-related tweet or non-software-related tweet. They then used Support Vector Machine (SVM) to train a model from a set of labeled tweets and applied the model on another set of tweets to identify each of them as software related or not [37]. In a later work, Tian et al. analyzed software microblogger behaviors using a dataset that contains more than 13 million tweets posted by 42,000 microbloggers [52]. An analysis of popular topics in Twitter was

performed in [43]. In [16], the authors explored what Twitter users say about software applications. A technique to extract informative URLs related to software engineering was proposed in [44].

Researchers have also built tools to support developers to better use Twitter for their day-to-day work [1, 37, 42]. Achananuparp et al. built a tool that visualizes trends based on a number of software-related tweets [1]. Sharma et al. proposed NIRMAL, which builds a language model based on the publicly available Stack Overflow data, and use it to compute a likelihood score of a tweet being software related or not [42]. The usages of various other social media channels have also been researched lately. Storey et al. surveyed software developers on how they use social media channels to communicate and collaborate, and present insights into the challenges faced by developers while using the channels [48, 49]. Macleod et al. explored how developers use YouTube for documenting and sharing programming knowledge [28]. Ponzanelli et al. developed a tool that extracts relevant fragments from software development video tutorials [35]. Lin et al. recently studied how developers use *slack*[1] to help them in their software development activities [24]. Our work recommends specialized software gurus, which is not considered in these prior works.

Past studies have explored the problem of recommending experts to certain tasks, e.g., bug triaging [3, 4, 27, 62, 64], question answering [9, 10, 17], and so forth. Our approach is related but different from these existing techniques. First, Twitter data contains a lot of noise since the topics of information shared on Twitter may include software engineering, politics, economy, idle chatter, and more. Second, Twitter data has unique features; for example, Twitter data contains information about people who follow a user or are followed by the user, how much a tweet is liked and/or shared (retweeted) by others, and so forth. Not utilizing these unique features is a missed opportunity.

The closest work to ours is that by Pal and Counts [33], which proposes an approach to rank Twitter users given a particular topic (represented by one or more keywords). They first extract a set of features from each Twitter user, i.e., the 10 content features that are considered in this work. Next, they generate a feature vector to represent each Twitter user who has posted tweets that contain the input keywords, and use them to cluster the Twitter users. The most likely cluster is selected and members in this cluster are deemed as experts. The members are also ranked to create a ranked list of domain experts. In this work, we extend the work by Pal and Counts by considering not only content features but also network, profile, and GitHub features. We also leverage a two-step classification algorithm to improve the accuracy of Pal and Counts's clustering approach.

There also exist systems such as Klout [38] that provide influence scores for Twitter users with respect to a queried domain. These scores can be interpreted as a measure of expertise of a Twitter user over a given domain. In this work, we compare our two-step classification algorithm with Klout and find that it performs better than Klout (see Section 6).

3 WHO TO FOLLOW: DEVELOPERS' PERSPECTIVE

To guide and motivate the design of our automated recommendation system, we conducted an open-ended online survey with developers who have already made use of Twitter in their software development activities. We investigated the kinds of users they would like to follow on Twitter. The survey details are described below.

Survey Design and Analysis: The primary objective of our survey is to understand what categories of Twitter users software developers like to follow. To understand this, we designed an open-ended survey. Our survey consisted of three key questions:

[1] https://slack.com/.

- The first question asks whether a respondent develops software systems and uses Twitter in his or her software development activities. People who have not developed software systems or not used Twitter in their software development activities may not have sufficient background to respond to our survey. This question aims to validate the reliability of the answers that we receive for the subsequent questions.
- The second question asks a respondent for his or her years of experience as a software developer (less than 5 years, 5 to 10 years, or more than 10 years).
- The third question asks a respondent to indicate the types of Twitter accounts they like to follow for the purpose of helping them in software development activities. This question was open ended and the respondent was asked to give a text description of accounts they follow or would like to follow.

We then analyzed the responses provided by developers using grounded theory methodology [12, 39]. Specifically, we used open card sort [18] in order to develop categories of Twitter accounts that software developers like to follow. The first two authors of the paper are involved in the open card sort process. Our card sorting process has three phases. In the *preparation* phase, each response is read, and cards are created based on the user responses. Some users mentioned more than one type of account they would like to follow; for such cases, we create multiple cards. Next, in the *execution* phase, all the cards are clustered into meaningful groups. Finally, in the *analysis* phase, based on the clusters we get from the last phase, we formed a higher-level theme and categories to come up with the final categories. In the card sort process, we ignore responses such as "I look for accounts that are insightful or informative" as they are too general to be put into a specific category. Additionally, we merge categories that are mentioned by less than three respondents into a special category *Others*. The open card sort process was performed together by the first two authors. Our process is similar to the negotiated agreement technique described in [8]. As the card sorting has been performed together, there is no interrater agreement number. Many previous studies involving card sorting have also followed a similar process [2, 20, 21, 26, 46].

Survey Participants: We targeted software developers who are present on Twitter. Following [1, 42, 43, 52], we collect a set of 161,067 Twitter users who are potentially interested in software development—see Section 6.1 for details. Next, we identify from this set a subset of users who satisfy two criteria: (1) they are recently active (i.e., those who had posted tweets after February 2017), and (2) they allow anyone to send them Twitter direct messages.[2] Users who have not been recently active on Twitter may not respond to our survey requests—and thus the first criterion. The second criterion is there since we need to use the Twitter direct messaging service to connect to our potential survey participants. This service allows us to send a detailed personalized message to users, which would not have been possible if we had contacted the users by sending tweets as they are limited to 140 characters. After creating this subset of users, we randomly select users from it to contact. The first author of the article has sent personalized Twitter direct messages to hundreds of these users, requesting them to fill out the survey. In total, we have contacted 213 developers, out of which 38 developers responded back by filling out the survey. This translates to a response rate of 17.84%. We discarded the responses of two respondents since they did not use Twitter in their software development activities (i.e., they respond with a "No" for the first survey question). We performed an open card sort on the remaining 36 responses.

After the card sort, in order to decide whether the survey responses are adequate, we checked if the responses have reached saturation. According to Strauss and Corbin [50], sampling for a survey can be terminated when collecting new data does not generate any new information.

[2] https://support.twitter.com/articles/14606.

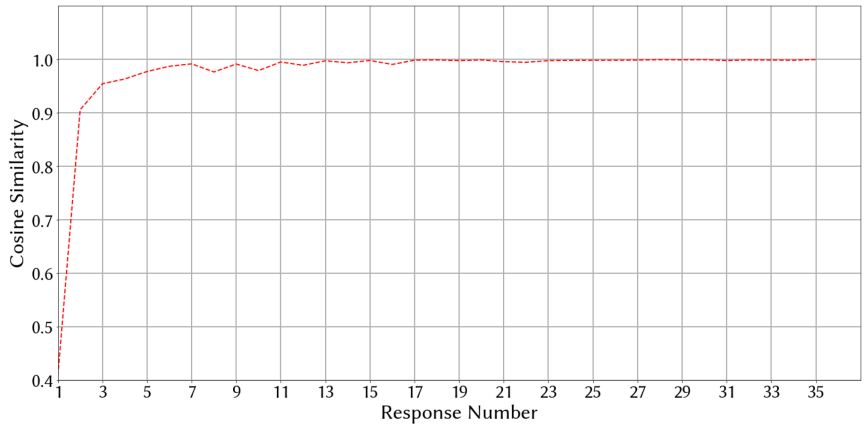

Fig. 1. Graph showing saturation of $CosSim_n$ score.

During the survey, we observed that after we got about 25 responses, new responses were not leading to any new insights or information. This observation suggested that theoretical saturation had been reached, so we decided to stop the survey and perform the card sort. We had already received 36 responses by the time we stopped the survey, so we went on to perform the card sort on all of the 36 responses. To further validate and check for saturation, we used the following steps. We first represented the n^{th} survey response as a vector R_n of size equal to the number of categories we developed through card sort. Each element of R_n represents a category, with the default value of the element being 0. The element corresponding to a category is assigned a value of 1 if the response mentioned the category. Then, for the n^{th} response, we calculated the average mean response for the first n responses A_n as follows:

$$A_n = \frac{\sum_{i=1}^{n} R_n}{n}.$$

The intuition behind using the vector A_n is to validate if getting a new response helps us to get any new information (category in our case). The A_n vector does not change much when the new response does not mention new information (or category). This can be captured by measuring cosine similarity between subsequent vectors A_n and A_{n+1}. After computing A_n for the 36 valid responses, we then compared pairs of vectors A_n and A_{n+1} using cosine similarity [30]. The cosine similarity $CosSim_n$ between the n^{th} and $(n+1)^{th}$ responses is computed as

$$CosSim_n = \frac{A_n \cdot A_{n+1}}{\|A_n\| \|A_{n+1}\|}.$$

In the above equation, $\cdot$ is the dot operation between vectors and $\|A_i\|$ is the size of vector A_i. Saturation can be observed when the value of $CosSim_n$ stabilizes and does not change much when a new response is added. The value of $CosSim_n$ is shown in Figure 1. We can note that $CosSim_n$ stabilizes after the $23rd$ response. So based on this observation, we decided not to send out any further requests to developers for filling out our survey.

Survey Results: By analyzing the responses to the first question of the survey, we found that 94.73% of the respondents (i.e., 36) have developed software systems and use Twitter for their software development activities.

After performing the open card sort on the responses provided by the 36 respondents, we were able to identify five prominent categories apart from *Others*. These categories are shown in Table 1.

Table 1. Categories of Twitter That Users/Accounts Developers Like to Follow on Twitter

Code	Category
I	Accounts of domain experts (includes well-known developers, library & framework authors, etc.)
II	Accounts that provide technology related news
III	Accounts of software organizations/companies/firms related to a particular domain
IV	Accounts of CTOs/CEOs of software/technology companies of a particular domain
V	Accounts of software frameworks/tools/libraries related to a particular domain
VI	Others

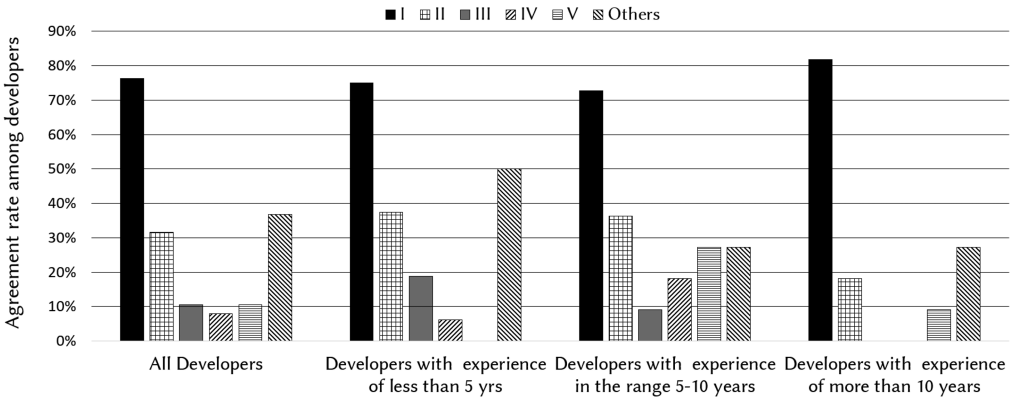

Fig. 2. Infograph displaying what types of Twitter accounts developers across different experience levels prefer to follow. For descriptions of categories I through V, please refer to Table 1.

Figure 2 shows the percentage of our survey respondents who mention a particular category in their response to the third question of our survey. From the figure, we can note that only one category, i.e., accounts of domain experts, is preferred by more than 70% of respondents. The choice of this category is consistent among developers across all experience levels. Based on this result, in the rest of this article, we focus on building an automated tool to recommend domain experts who generate specialized domain contents that others can benefit from (i.e., specialized software gurus) and evaluate our results by asking people to label whether a recommended Twitter account belongs to such domain experts. We do not consider the other five categories as a substantial majority of respondents (62.50% to 100%) are not interested in following users belonging to them.

4 DOMAIN-SPECIFIC CHARACTERIZATION OF TWITTER ACCOUNTS

In this section, we describe the features that we use to characterize a Twitter user (i.e., a registered account on Twitter) given a particular specialized domain of interest. In this work, a domain corresponds to a software engineering concept of interest and is represented by a keyword. In particular, we consider two programming language keywords (i.e., JavaScript and Python) and two operating system keywords (i.e., Android and Linux). We pick these keywords as they are popular, well represented in our dataset, and known well to participants we hired for labeling experts.

We consider four feature families *Content*, *Network*, *Profile*, and *GitHub* , each of which is described in the following subsections.

4.1 Content Features

Content features characterize how often a Twitter user generates tweets about a specialized domain or topic of interest and the impact of his or her tweets on other users. Users who frequently post about a domain are likely to have expertise in the given domain. Among such users, those who interact frequently with other domain-related users are more likely to be specialized software gurus.

We reuse a set of features first proposed by Pal and Counts to recommend domain experts on Twitter [33]. Before we present the features, we need to first introduce some feature components and terminology related to them. These feature components are then combined to arrive at scores for content features.

Terminology: Given a particular domain that is represented by a keyword, e.g., Python, we define the following concepts:

- *Domain-related tweets* are tweets that contain the representative keyword.
- *Domain-related hashtag* is a word that starts with the # symbol and contains the representative keyword, e.g., #Python for keyword Python.
- *Domain-related Twitter users* are Twitter users who have posted 10 or more domain-related tweets.

The tweets generated by a user can be categorized into the following three categories:

- *Conversational tweets (CTs)* are tweets that mention at least one Twitter user.
- *Retweeted tweets (RTs)* are tweets that are originally generated by someone else and the Twitter user copies or forwards them in order to spread the information to his or her followers.
- *Original tweet (OTs)* are the non-RT and -CT tweets that are produced by a Twitter user.

Based on the above concepts, Table 2 presents feature components that can be calculated for each Twitter user. These feature components are used to construct more complex content features later. The concept of "friend" is used to calculate G2 and G4. A user A and user B are friends of each other if both A and B follow each other, and thus get automatically subscribed to each other's tweets.

Features: We consider a total of 10 content features as proposed in [33]. These features are based on the feature components introduced in Table 2. All of them are calculated for each user with respect to a particular domain. We further subcategorize the content features into categories of *Content Strength* and *Content Popularity*. We describe the subcategories and their respective features below:

Content Strength: The features under this category measure how closely related the content generated by a Twitter user is to a given domain.

- *Topical Signal*: Topical Signal (TS) estimates how much a user u is involved with the domain d irrespective of the types of tweets posted by him or her. The TS score of a Twitter user u for a domain d is defined as

$$TS(u,d) = \frac{OT1(u,d) + CT1(u,d) + RT1(u,d)}{\#AllTweets(u)}.$$

  In this equation, $\#AllTweets(u)$ refers to the total number of tweets generated by user u whether or not they are domain-related tweets. This feature can take values in the interval [0,1].

Table 2. A List of Feature Components

Component Name	Component Description
OT1(u,d)	Number of original tweets related to domain d posted by a user u
OT2(u,d)	Number of URL links shared in tweets related to domain d posted by a user u
OT3(u,d)	Number of hashtags related to domain d used in tweets posted by a user u
CT1(u,d)	Number of conversational tweets related to domain d posted by a user u
CT2(u,d)	Number of conversational tweets related to domain d where conversation is initiated by a user u
RT1(u,d)	Number of times a user u retweets tweets related to domain d of other users u
RT2(u,d)	Number of unique original domain-related tweets of a user u that are retweeted by other domain-related users, where domain is d
RT3(u,d)	Number of unique domain-related users who retweet original domain-related tweets of a user u, where domain is d
M1(u,d)	Number of mentions of other domain-related users by a user u in his or her domain-related tweets, where domain is d
M2(u,d)	Number of unique domain-related users mentioned by a user u in his or her domain-related tweets, where domain is d
M3(u,d)	Number of mentions of a user u by other domain-related users in their domain-related tweets, where domain is d
M4(u,d)	Number of unique domain-related users mentioning a user u in their domain-related tweets, where domain is d
G1(u,d)	Number of domain-related followers of a user u, where domain is d
G2(u,d)	Number of domain-related friends of a user u, where domain is d
G3(u,d)	Number of domain-related followers generating domain-related tweets after a user u generated a domain-related tweet, where domain is d
G4(u,d)	Number of domain-related friends generating domain-related tweets before a user u generates a domain-related tweet, where domain is d

- *Signal Strength*: Signal Strength (SS) indicates how strong a user's topical signal is, such that for a true authority this score should approach 1. This feature can take values in the interval [0,1]. The SS score of a Twitter user u for a domain d is defined as

$$SS(u,d) = \frac{OT1(u,d)}{OT1(u,d) + RT1(u,d)}.$$

- *Nonchat Signal*: Nonchat Signal (NCS) captures how much a user posts on the domain and how much he or she digresses into conversations with other users. The NCS score of a Twitter user u for a domain d is defined as

$$NCS(u,d) = \frac{OT1(u,d)}{OT1(u,d) + CT1(u,d)} + \lambda \frac{CT1(u,d) - CT2(u,d)}{CT1(u,d) + 1}.$$

  As discussed in Pal and Counts [33], the intuition behind adding the second fraction in the above formulation is to discount cases when the account did not start the conversation but simply replied back out of courtesy. This is desirable as we wish to find real experts rather than organizations who are somewhat more social. The second fraction accounts for such cases. We have used the λ value as 0.05, as also used by Pal and Counts [33]. The second

fraction contains 1 in the denominator to account for cases where CT1(u,d) is 0. This feature can take values in the interval (0,1).

- *Self-Similarity*: Self-Similarity (SelfS) indicates how similar is a user's recent tweet w.r.t. to his or her previous tweets. To compute SelfS for a user u, first, from each tweet i of the user u, commonly used words are removed based on a stop word list.[3] Then each tweet i is represented as a vector of words s_i that contains the remaining nonstop words. Then, the similarity S between two tweet vectors s_i and any previous tweet s_j is calculated as follows:

$$S(s_i(u), s_j(u)) = \frac{|(s_i(u) \cap s_j(u)|}{|s_i(u)|}.$$

The self-similarity score for a user u is computed as the average similarity scores for all pairs of tweets:

$$SelfS(u) = \frac{2 \cdot \sum_{i=2}^{n} \sum_{j=1}^{i-1} S(s_i(u), s_j(u))}{(n-1)n}.$$

In this equation, n is the total number of tweets generated by u irrespective of the domain. This feature can take values in the interval [0,1].

- *Link Rate*: Link Rate (LR) for a user u considering domain d is the ratio of the number of URL links a user u shared in his or her domain-related tweets to the total number of domain-related tweets made by user u:

$$LR(u, d) = \frac{OT2(u, d)}{OT1(u, d)}.$$

Since a tweet is short and deep technical contents cannot be elaborated in 140 characters, a higher LR score might improve the likelihood of a topic-related tweet being useful to other developers. Twitter has a limit of 140 characters per tweet and each URL shared consumes 23 characters, so a tweet can at the maximum contain five URL links. Thus, this feature can take values in the interval [0,5].

- *Domain-Hashtag Rate*: Domain-Hashtag Rate (HR) is similar to link rate, but it considers the proportion of domain-related tweets that contain a domain-related hashtag. The HR score of a Twitter user u for a domain d is defined as

$$HR(u, d) = \frac{OT3(u, d)}{OT1(u, d)}.$$

Hashtags in a tweet are created by adding "#" before any character other than a space or punctuation. So any hashtag will at least contain two characters (including the "#"). Twitter has a limit of 140 characters per tweet, and if a single character preceded by "#" is used as a hashtag, then a tweet can contain a maximum of 47 hashtags (94 characters for hashtags and 46 for spaces in between hashtags). So, this feature can take values in the interval [0,47].

Content Popularity: The features under this category measure how popular and impactful is the domain-related information generated by a user.

- *Retweet Impact*: Retweet Impact (RI) indicates the impact of the contents generated by the user. The RI of a Twitter user u for a domain d is computed as

$$RI(u, d) = RT2(u, d) \cdot log(RT3(u, d)).$$

The retweet impact is primarily captured by RT2, which measures how many times a user u has been retweeted. However, for some users the values of RT2 can be high just because

[3] http://www.ranks.nl/stopwords.

some of their devoted followers always retweet the content. To dampen the impact of such users, multiplication by the logarithm of RT3 is done, as RT3 only captures the unique followers who retweet content of a user. This feature can take values in the interval [0,∞).

- *Mention Impact*: Mention Impact (MI) indicates how much an account is mentioned with regard to the domain of interest. The MI score of a Twitter user u for a domain d is defined as

$$MI(u,d) = M3(u,d) \cdot log(M4(u,d)) - M1(u,d) \cdot log(M2(u,d)).$$

  MI is measured as a difference of two components mentioned below:
  * The first component is a product of M3 and logarithm of M4. Mainly, M3 gives a good estimate of this component. However, in order to account for mentions being received from people known to a user, M3 is multiplied by the logarithm of M4. As M4 consists of only unique users, its logarithm helps to dampen the impact of M3.
  * The second component is a product of M1 and logarithm of M2, which measures how much a user is mentioning other users in Twitter. Again, the logarithm of M2 is used to dampen the impact of people frequently mentioned by the user. Sometimes a user may also receive mentions back only because of the fact that they mention others. To account for this factor, we need to subtract the second component (which estimates how often the user mentions others) from the first component.

  The above steps ensure that the MI we calculate for a user is based on his or her merit and not as a result of him or her mentioning other users. This feature can take values in the interval [0,∞).
- *Neighbor Score*: Neighbor Score (NS) captures the raw number of domain-related users for a domain d around a user u. The network score of a user u for a domain d is computed as

$$NS(u,d) = log(G1(u,d) + 1) - log(G2(u,d) + 1).$$

  Instead of using the absolute values of G1 and G2, their logarithms have been used here to avoid issues with clustering as the distribution of G1 and G2 follows a long-tail distribution [33]. This feature can take values in the interval [0,∞).
- *Information Diffusion*: Information Diffusion (ID) estimates how much influence is diffused by the user in its network. We define the ID score of a Twitter user u for a domain d as

$$ID(u,d) = log(G3(u,d) + 1) - log(G4(u,d) + 1).$$

  Similar to NS, logarithms have been used here. This feature can take values in the interval [0,∞).

4.2 Network Features

In Twitter, one user is connected to other users via the *follow* relationship. For each Twitter user, we can thus form a network of other users that are connected to it via this follow relationship (either directly or indirectly). In this network, we can estimate the importance of a user in the network. A software guru who shares many gems of knowledge with others is likely to be followed by many other developers who benefit from his or her microblogs and thus is expected to have a high importance score among other users in the network.

To capture the above-mentioned intuition, we create a network for each domain where nodes correspond to domain-specific users and edges correspond to the follow relationships among these users. The edges in our network are directed; e.g., an edge from user A to user B in our graph means that the user A follows user B on Twitter. We then evaluate the importance of each user in this network.

To measure the importance of a user, we build upon various studies in web and social network mining communities, which have proposed various metrics [6, 13, 32, 61]. We use some of the centrality indicators proposed in [6, 61], which are widely used in network analysis and graph theory to identify the most important nodes and vertices in a graph or a network. We also use *PageRank* proposed by Page et al., which gives authority scores of important nodes in a network [32]. Intuitively, software domain experts are typically known by many people in the domain and expected to interact with others. Thus, it is expected that the nodes representing experts would be important and centrally located. The network features have been further categorized into *Centrality Scores* and *Absolute Scores*. We describe the subcategories and their respective features below. Using the features mentioned below would help in identifying the experts.

***Centrality Scores*:** Features in this category are metrics based on research in social and network mining communities and they measure how central (important) a node (user) is in a network (Twitter).

- *Betweenness*: Betweenness is defined based on the number of shortest paths from all nodes to all others that pass through a node. A high score for this measure means that very often this node (equivalent to a user in the Twitter network) serves as a *bridge* between other nodes. We believe that many software gurus act as *knowledge brokers* and help to facilitate information flow between various parties. Singer et al. also observe that thought leaders also mention and retweet contents generated by others [45]. The betweenness score helps us to identify such broker nodes in the Twitter network and thus we have used it as a network feature in this work.
- *Closeness*: Closeness is defined as the reciprocal of the average shortest distance of a node to all the other nodes in a network. The intuition behind this feature is that gurus are expected to be directly or indirectly connected to a large number of other users a few hops (edges) away, and hence on average are closer and easily reachable by others. The closeness scores help us to identify such potential gurus.
- *Degree Centrality*: Degree Centrality for a user u is the ratio of users to which it is connected to the total users in the network. The number of users connected to a user u includes the users who follow u and the users who are followed by u. A user who is a domain expert in Twitter generally has a large number of followers, resulting in a relatively large value of this feature.
- *OutDegree Centrality*: OutDegree Centrality for a user u is the ratio of the number of other users it follows to the total number of users in the network. Intuitively, experts on Twitter have a large number of followers but do not follow a large number of accounts, so the value of the OutDegree Centrality feature is expected to be low for experts.
- *PageRank*: PageRank (PR) is a node importance measurement method proposed by Page and Brin [32]. The PR algorithm computes a probability to represent the likelihood of a *walker* arriving at a particular node by randomly traversing edges in a network. The PR algorithm runs iteratively. At iteration i, the PR score of a node u is defined as follows:

$$PR(u, i) = \frac{1 - d}{N} + d \sum_{v \in B(u)} \frac{PR(v, i - 1)}{|L(v)|}.$$

  In the equation, d is the probability that a random walker continues to visit other nodes (aka the *damping factor*), N is the number of nodes in the network, $B(u)$ refers to the set of nodes that link to u, and $L(v)$ is the set of nodes that v links to.

  We use the PageRank method mentioned above to measure the importance of a user in a Twitter network, considering the importance of other users. Intuitively, a user that is

followed by many credible users is more likely to be credible. Highly credible users are likely to be software gurus who are followed by possibly many other gurus, or at least credible Twitter users who are highly interested in software engineering contents, in a particular domain of interest.

***Absolute Scores*:** Features in this category are based on the absolute number of users who follow or are followed by a user on Twitter.

- *Followers*: This feature indicates the number of people who follow a user on Twitter. If a user u has a high number of followers, it intuitively means that many other users are interested in the tweets generated by the users. Such users are expected to be highly popular and generally have a high probability of being experts in some domain.
- *Followed*: This feature indicates the number of people followed by a user on Twitter. If a user u follows a huge number of other users intuitively, it is expected to be not of an expert or human, as generally a single person cannot comprehend the information from tweets generated from a huge number of users they follow. Most of the times such users represent some organizational or bot accounts that are interested in monitoring the information generated from other users. Thus, the value of this feature can be an important factor in discerning domain experts.
- *NExpertFollowers*: This feature indicates the number of experts of a particular domain who follow a user. If a user u is followed by a lot of users who are experts in a particular domain, then most likely the user u will also be an expert in the domain. Thus, this feature value can be an important signal in finding experts in a particular domain.
- *NExpertsFollowed*: This feature indicates the number of experts of a particular domain followed by a given user u. A user u who follows a large number of experts of a particular domain is expected to be a user related to a domain. This feature when combined with other features should strengthen our approach in order to find users related to a particular domain.

4.3 Profile Features

A Twitter user can specify his or her biodata and include a reference to his or her webpage in his or her Twitter account. This information can help us to better characterize a Twitter user. Keywords such as *developer*, *architect*, *consultant*, and so forth in the biodata and webpage of users can help to identify software experts or gurus among other domain-related users. On the other hand, keywords such as *recruiter*, *headhunter*, and so forth help to identify and eliminate accounts related to hiring firms. These accounts are not preferred by most developers as discussed in Section 3.

To collect information from a Twitter user's biodata and webpage, we perform three steps: biodata and URL extraction, webpage preprocessing, and text preprocessing. In the first step, we process information from a Twitter account to extract the user's biodata and the URL to his or her webpage (if available). In the second step, if the URL to a user's webpage is specified, we download the webpage and extract textual contents from the webpage using a Python package called *BeautifulSoup*.[4] The Python package will remove HTML-related keywords and scripts that exist in the downloaded webpage. In the third step, we perform standard text preprocessing on the biodata and the webpage text, which includes the following substeps:

[4] http://www.crummy.com/software/BeautifulSoup/.

(1) *Tokenization:* We split the biodata/webpage text into tokens, where each token corresponds to a word that appears in the text.
(2) *Stop Word Removal:* We remove common English stop words, such as "is," "are," and so forth, since they appear very often and thus have little discriminative power. We use the list of English stop words provided on http://www.ranks.nl/stopwords.
(3) *Stemming:* We reduce a word to its root form (e.g., "reading" and "reads" are both reduced to "read") using a popular stemming algorithm, i.e., Porter stemmer [36]

In the end, for each user, we construct two feature vectors: one to represent his or her biodata and the other to represent his or her webpage. Each feature corresponds to a preprocessed word that appears in the biodata (or webpage), and the value of the feature is the number of times the word appears in the biodata (or webpage). We call the biodata feature vector as *Biodata* and the webpage feature vector as *Webpage*. These feature vectors are converted into four probabilities that represent the likelihood of a Twitter user being a specialized guru, and the process is discussed in detail in Section 5.2. We denote the four probabilities as *PosBio*, *NegBio*, *PosWeb*, and *NegWeb*. Apart from the above four probability scores, there are a few more profile-related features that are mentioned below:

- *IsVerified*: *Verified* accounts on Twitter represent accounts maintained by users who are popular in key interest areas such as music, sports, and so forth and whose authenticity has been confirmed.[5] A verified account related to a software domain and which is human also has a very high probability of being an expert in the domain.
- *AccountAge*: This feature indicates how long the user has been present on Twitter. A user who is present on Twitter for a long period of time and also generates domain-related tweets is likely be an expert developer.
- *CosSimWeb*: This feature measures the cosine similarity between the keyword representing the domain of interest and the *Webpage* feature vector. Users who have more domain-related text on their webpage are expected to be closer to the domain.
- *CosSimTweetText*: This feature measures the cosine similarity between the keyword representing the domain of interest and the text of all the original tweets made by the user. Users who tweet more on a particular domain have a higher probability of being an expert. This score is expected to be higher for such users.

4.4 GitHub Features

Some Twitter users include links to their GitHub profiles in their webpages. GitHub is one of the popular code and repository holding websites, having over 21.1 million repositories held by over 9 million users [14]. The presence of a GitHub account and high activity in GitHub can be important factors in identifying software experts. In this work, we use the following five basic GitHub features:

- *IsGhMentioned*: This feature indicates whether a Twitter user includes a link to his or her GitHub profile in his or her webpage. Intuitively, a software expert will want to publish a link to his or her GitHub profile on his or her webpage to highlight his or her work and possibly to find interested people to join the projects he or she is championing on GitHub. A newbie or a nonexpert developer is likely not to have a GitHub profile, and even if he or she has one, he or she may not have any/many projects to display or promote. Thus, newbies are less likely to highlight their GitHub profiles on their webpages. We set the value of this

[5] https://support.twitter.com/articles/119135.

feature to 1 if a valid GitHub profile link is present in a Twitter user's webpage; otherwise, it is set to 0.

- *GhFollowers*: This feature indicates the number of people who follow a user in GitHub. The more followers a user has, the more popular the user is, and thus the user has a higher likelihood of being an expert. This feature is assigned a value of 0 if IsGhMentioned = 0.
- *GhRepos*: This feature indicates the number of public repositories owned by a user in GitHub. More repositories implies that the user has worked on more projects, and thus this feature can be a good way to measure the expertise of the user. This feature is assigned a value of 0 if IsGhMentioned = 0.
- *GhGists*: This feature indicates the number of public Gists shared by a user in GitHub. Gists in GitHub are a way for developers to share useful code snippets or scripts. They are different from GitHub repositories, which are generally entire projects in themselves. A user having a large number of public GitHub Gists can be taken as an indicator of their experience in creating reusable solutions for common tasks or problems. It also suggests their willingness to share such information with other fellow developers. This feature can be a good way to find experienced developers who are also willing to share their experience with other developers. This feature is assigned a value of 0 if IsGhMentioned = 0.
- *GhUserType*: This feature indicates the type of GitHub Account. GitHub accounts can be of various types, such as individual accounts or those of organizations. This feature is assigned a value of 1 if the values returned by account type is "User"; otherwise, the feature is assigned a value of 0.

5 SOFTWARE GURU RECOMMENDATION

In this section, we first introduce the overall architecture of our prediction approach. We then describe in detail the key steps in the approach.

5.1 Approach Overview

Figure 3 shows the overview of our approach, which contains five major steps (candidate set creation, training set creation, feature extraction, classifier construction, and classifier application). Our approach takes as input a keyword specifying a domain of interest and users in Twitter and eventually produces a set of specialized software gurus.

In the first step, we select Twitter users that are potentially interested in software development from hundreds of millions of users. This helps us reduce the search space of finding specialized software gurus. We follow the approach used in [1, 42, 52], wherein initially we create a seed list of popular Twitter users who are software developers, by collating developers who are mentioned on technical blogs. We then expand this list by using the follow links of users present in the seed list. Next, as we need to find users who are related to a domain, we filter Twitter users who post fewer than 10 domain-related tweets for the month of December 2016. This gives us a candidate set of specialized software gurus related to a domain. The process is described in detail in Section 6.1.

In the second step, among the candidates identified in the first set, we manually label some of them as specialized software gurus or other users (details in Section 6.1), and this set of labeled users forms the training set. In the third step, we extract various features (i.e., content, network, profile, and GitHub features) described in Section 4 for all users in the candidate set. In the classifier learning step, the features of the users in the training set are used to learn a discriminative model (aka a classifier) that is able to differentiate specialized software gurus and other users based on their features. In the classifier application step, we apply the classifier on other candidate users who are not in the training set and predict those who are specialized software gurus. We describe the detail of our classifier construction step in the next subsection.

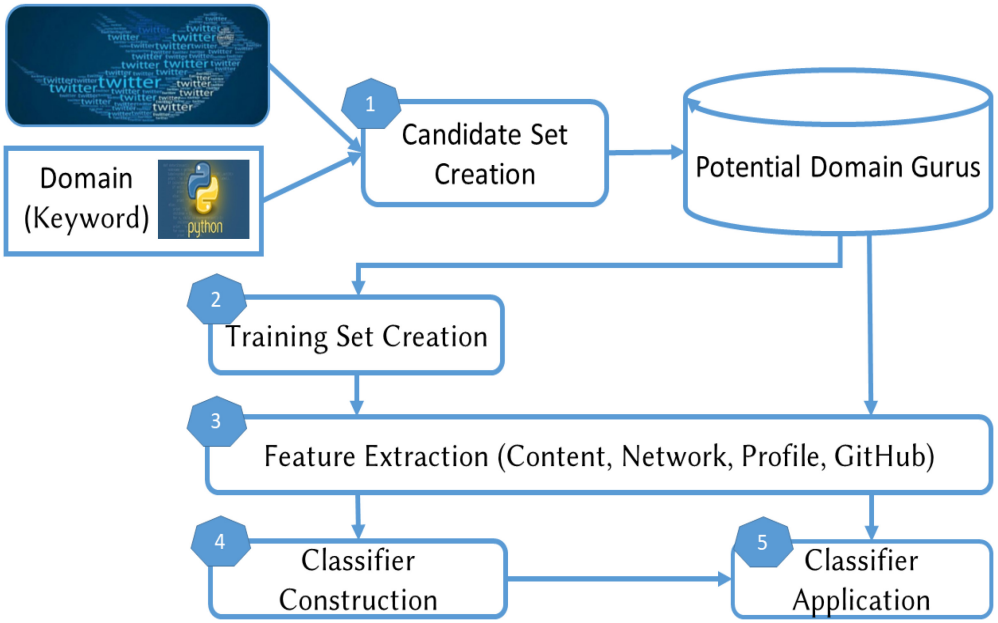

Fig. 3. Framework of our recommendation system.

5.2 Classifier Construction

To construct a classifier, our approach first processes thousands of profile features and then merges them with the other features to construct a unified discriminative model. We describe the detailed process below.

Processing Profile Features. Different from content, network, and GitHub features, the profile features based on biodata and webpage are not metrics but thousands of preprocessed words. The number of these profile features is large as compared with the number of features from the other families. Therefore, to make profile features based on biodata and webpage more comparable to other features, we convert these profile features into four probabilities that represent the likelihood of a Twitter user being a specialized guru. These four probabilities include the probability of a Twitter user to be a specialized guru given his or her biodata (i.e., $P(Guru|Biodata)$), the probability of a Twitter user to be not a specialized guru given his or her biodata (i.e., $P(\neg Guru|Biodata)$), the probability of a Twitter user to be a specialized guru given his or her webpage (i.e., $P(Guru|Webpage)$), and the probability of a Twitter user to be not a specialized guru given his or her webpage (i.e. $P(\neg Guru|Webpage)$). We denote the four probabilities as *PosBio*, *NegBio*, *PosWeb*, and *NegWeb*, respectively.

To obtain the four probabilities, we train two text classifiers from the biodata and webpages of users in the training set. We then apply these classifiers on all candidate users to generate the four probabilities for all users. By default, we use Naive Bayes Multinomial (NBM) as the default classifier to transfer profile features to the four probabilities. The NBM classifier is fast and has shown its discriminative power in similar situations, e.g., [63].

Constructing a Unified Discriminative Model. After we have processed the profile features, we combine the four probabilities with the 10 content features, four other profile features, nine network features, and five GitHub features to characterize a Twitter user. We then take the features of

users in the training data to learn a unified discriminative model (a classifier) that can differentiate specialized gurus from other users based on all of their features. After combining all the features, we again apply the NBM on the 32 features from the four families (i.e., content, network, profile, and GitHub).

6 EXPERIMENTS AND RESULTS

In this section, we first describe our dataset and experiment settings. Next, we introduce our research questions and present our experiment results that answer each of the research questions. At the end of this section, we present the threats to validity.

6.1 Dataset

The input dataset for our experiments is a set of a few million tweets that we collected in December 2016. To collect these tweets, we first created a seed list of popular Twitter users of software developers. To create this list, we first collected 100 Twitter users who are also popular software developers as mentioned in a technical blog;[6] this list of seed users was used by previous studies [1, 42, 52]. As this list is quite old, we also collected Twitter users who are popular software developers as mentioned in several other more recently published technical blogs.[7,8,9,10] From these blogs, we are able to extract 48 unique users. These 48 users were then merged with the previous 100 users, which results in a final set of 139 users (after removing duplicates), which we refer to as *uSeed*.

We then expanded the seed set by adding Twitter users who follow or are followed by at least N of the seed users in *uSeed*. In Twitter, if a user B follows another user A, it means any tweets published by A will be available to B. If B follows N users in *uSeed*, intuitively B is likely to be interested in software engineering content. Also, in case B is followed by N already identified software developers present in *uSeed*, then B has a very high probability of being a user producing content related to software engineering. We refer to this expanded set as *uBase*, and it contains 161,067 users. In our study, we pick the value of N to be 5. We then collect tweets that are generated by the users in *uBase* over a 1-month period (i.e., December 1–31, 2016). We were then able to download 5,517,878 tweets generated by 86,824 of the total 161,067 users in *uBase* for the month of December 2016.

The approach that we use in this article of using a seed network and extending it based on follow links helped us to expand our relevant user base (i.e., Twitter users who are likely to generate software engineering contents) quickly. An alternative way of doing this might be to search for LinkedIn pages, identify software developers based on their job titles, and search if their Twitter handles are mentioned in those pages. This may result in a cleaner dataset, since we are sure that those Twitter users are really corresponding to software developers. However, not all LinkedIn pages contain Twitter handles. Additionally, software developers have different job titles. Most importantly, LinkedIn restricts us from crawling its pages.[11]

We evaluated the effectiveness of our approach by recommending software gurus for four domains: JavaScript, Android, Python, and Linux. Javascript and Python are programming languages, while Android and Linux are operating systems. We chose these domains since among tweets in

[6] http://www.noop.nl/2009/02/twitter-top-100-for-softwaredevelopers.html.
[7] https://www.untapt.com/blog/2015/11/25/developers-to-follow-on-twitter/.
[8] https://www.thebalance.com/programmers-on-twitter-2072010.
[9] http://zartis.com/ten-software-developers-follow-twitter/.
[10] http://www.techworld.com/picture-gallery/social-media/people-all-developers-should-follow-on-twitter-3644265/.
[11] https://techcrunch.com/2016/08/15/linkedin-sues-scrapers/.

Table 3. Dataset Statistics

Dataset	Period	#Tweets	#TotalUsers	#FilteredUsers
All	December 2016	5,517,878	86,824	-
JavaScript	December 2016	27,466	9,369	293
Android	December 2016	20,655	6,951	247
Python	December 2016	11,074	3,710	127
Linux	December 2016	12,344	4,805	118

Twitter Username	SWPrac.	DomainPrac.	SWExp.	DomainExp.	TweetsHelpful	Domain	Link
kevinmarks	Y	Y	Y	Y	N	android	display
thej	Y	Y	Y	Y	Y	android	display
rtanglao	Y	Y	Y	CD	N	android	display
arstechnica	N	N	N	N	Y	android	display
wire	N	N	N	N	N	android	display
jrobertson	Y	CD	Y	CD	N	android	display

Fig. 4. Main page of our labeling system.

our dataset, these domains were well represented. Indeed, in our dataset, there are more JavaScript-related tweets than any other domain-related tweets. Another consideration was that we were easily able to find people to label the data as gurus and nongurus for these domains. Since a domain-related user that generates too few domain-related tweets may not be interesting to follow, as an additional step, we filter Twitter users who have tweeted fewer than 10 domain-related tweets in a month. We also chose only those users whose Twitter profile mentioned English as their preferred language. We show the total number of filtered domain-related tweets and Twitter users in Table 3. For these domain-related Twitter users we also crawled their biodata from their Twitter profiles and downloaded the websites whose URLs are mentioned in the users' Twitter profiles. Table 3 summarizes basic statistics of our dataset.

Next, we asked six PhD students majoring in Computer Science and two experienced software developers to label our dataset, which contains 293 JavaScript-related, 247 Android-related, 127 Python-related, and 118 Linux-related Twitter users. Each of the participants had more than 5 years of experience in programming and some experience in the respective technology domain whose users they labeled. The participants were hired by word-of-mouth approach and email requests, and none of them had any insights into how our algorithm works or the features that we used. For each domain, the data was labeled by two to three persons independently. A participant was assigned to a domain only if he or she had some experience in the domain whose users were to be labeled. In the labeling task, each labeler had to answer some questions with respect to each user in the given domain. Then, on the basis of the answers to these questions, it was determined if the user is an expert in the domain under consideration. After the data for a domain was labeled independently by the labelers, we computed the interrater agreement. For cases in which they disagree, the labelers sat down together to discuss and decide final labels.

To better support the labeling process, we provided a web-based labeling system for the participants. Figure 4 shows the main page of our labeling system, which contains a list of Twitter users who need to be labeled. For each user, the participant had to click the "display" button to enter an evaluation page. Figure 5 shows the evaluation page for a Twitter user. This page contained five parts: (I) user account name; (II) details from the user's Twitter profile, which include the user's

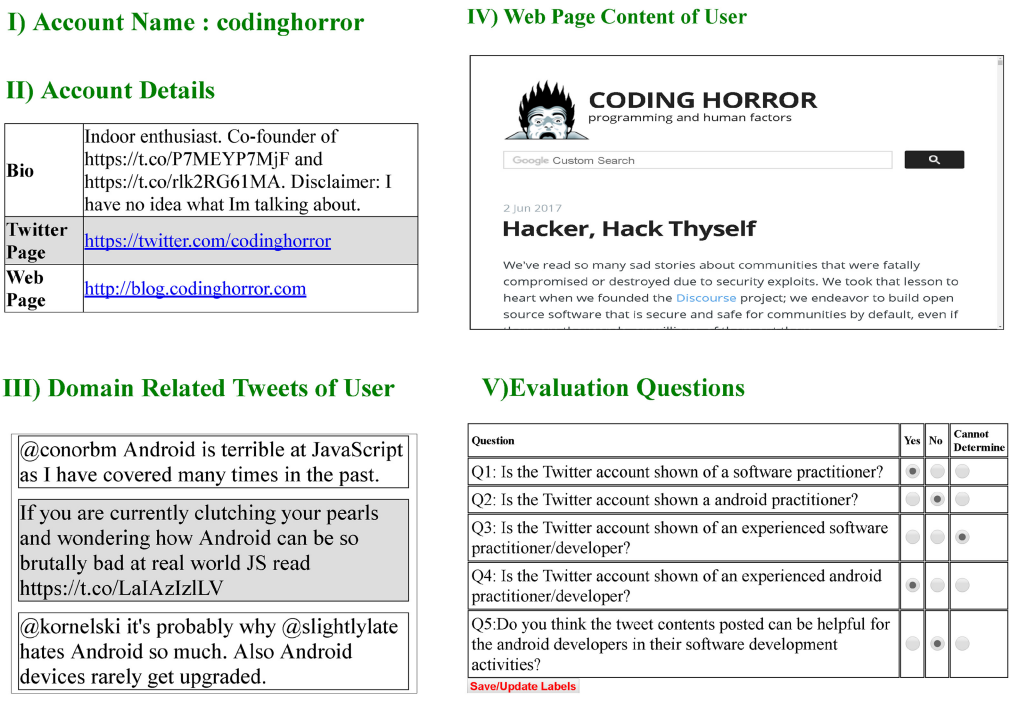

Fig. 5. Evaluation page for a Twitter user.

biodata; (III) all domain-related tweets that were posted by the user in our dataset; (IV) contents of the webpage whose URL is specified in the user's account profile; and (V) evaluation questions.

We asked participants to answer five questions in part V based on the information shown in parts I through IV. The first question asked a participant if the user shown on screen is a software practitioner. The second question evaluated whether the user shown is a practitioner in the particular domain of interest, e.g., if he or she is a JavaScript practitioner. The third question asked whether the Twitter user is an experienced software practitioner. Finally, the fourth question asked whether the Twitter user is an experienced practitioner on the particular domain of interest. The last question asked whether tweets posted by the user could be useful for developers who are working on the specific domain of interest. For each question, a participant needed to provide one of the three answers: "Yes," "No," or "Can't Determine."

The answers to questions 4 and 5 determined the label of a Twitter user (i.e., "Specialized gurus" or "Others"). For Twitter users for which both questions 4 and 5 were answered as "Yes," we labeled them as "Specialized gurus." These users are experienced developers in the domain of interest who post contents in Twitter that potentially benefit other developers in the same domain. For users who received an answer for question 4 as "Yes" and an answer for question 5 as "No," we labeled them as "Others." For users who received an answer to question 4 as "No," we labeled them also as "Others." We omitted the rest of the users from the final dataset, since their labels cannot be reliably determined.

The interrater agreement scores for answers to questions 4 and 5 over all domains are shown in Table 4. We used Cohen's Kappa [11] to measure interrater reliability for the labeling task. A Cohen's Kappa score less than or equal to zero is considered as no agreement, between 0.01 and 0.20 is considered as none to slight agreement, between 0.21 and 0.40 as fair agreement, between

Table 4. Interrater Agreement

Domain	Users	Q4		Q5	
		Cohen's Kappa	Agreement	Cohen's Kappa	Agreement
Javascript	293	0.61	substantial	0.51	moderate
Android	247	0.53	moderate	0.67	substantial
Python	127	0.41	moderate	0.47	moderate
Linux	118	0.25	fair	0.33	fair

Table 5. Number of Specialized Software Gurus

Domain	#Guru	#Others
JavaScript	98	87
Android	44	184
Python	26	65
Linux	38	72
All	206	408

0.41 and 0.60 as moderate agreement, between 0.61 and 0.80 as substantial agreement, and between 0.81 and 1.0 as almost perfect agreement [31, 59]. We can see from Table 4 that except for the Linux dataset, the agreement is at least moderate. For the Linux dataset, the agreement is still fair.

Table 5 shows the results of our labeling process after all the initial labeling and disagreement resolution. In the end, we have a total of 614 domain-related Twitter users who are labeled as "Specialized gurus" or "Others." About 33.55% of the total users in our dataset were labeled as gurus. The proportion of gurus is not very small as they are identified among Twitter users who post at least 10 domain-related tweets in a 1-month period, and whose labels can be reliably determined. For example, for the "Python" domain, initially a total of 3,710 Twitter users had posted at least one tweet having the keyword "Python." Out of these, only 127 users had posted at least 10 domain-related tweets. Further, during annotation, labels were reliably determined only for 91 of these users, out of which 26 were labeled as "Specialized gurus." Thus, the two steps of filtering and labeling result in an increased proportion of gurus in our final dataset. For the "JavaScript" domain, the number of "Specialized gurus" is more than 50% of the total users of that domain. This can be explained by the fact that "JavaScript" is currently the most popular programming language,[12] so the number of "JavaScript" gurus on Twitter is also expected to be more. We use these 614 users to evaluate the effectiveness of our approach in differentiating specialized domain gurus from other domain-related users.

6.2 Experiment Setting

Implementation Details: We use the implementation of Multinomial Naive Bayes[13] provided as part of sklearn [7].

Evaluation Metrics: We use three standard metrics, namely, precision, recall, and F-Measure, which have been used in many past studies, e.g., [53, 63]. They are calculated based on four possible outcomes of a Twitter user in an evaluation set: the user is a specialized software guru and he or

[12] https://insights.stackoverflow.com/survey/2017#technology.
[13] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes. MultinomialNB.html.

she is correctly predicted as such (true positive, TP); the user is not a specialized software guru, but he or she is wrongly predicted as a specialized software guru (false positive, FP); the user is a specialized software guru, but he or she is not predicted as such (false negative, FN); or the user is not a specialized software guru, and he or she is correctly predicted as such (true negative, TN). Based on these possible outcomes, precision, recall, and F-Measure are defined as:

Precision is the proportion of correctly predicted specialized software gurus among those predicted as specialized software gurus, i.e., Precision = $\frac{TP}{TP+FP}$.

Recall is the proportion of specialized software gurus that are correctly predicted as specialized software gurus, i.e., Recall = $\frac{TP}{TP+FN}$.

F-Measure is the harmonic mean of precision and recall, and it is used as a summary measure to evaluate if an increase in precision (recall) outweighs a reduction in recall (precision), i.e., F-Measure = $\frac{2\times Precision\times Recall}{Precision+Recall}$.

Evaluation Procedure: We apply 10-fold cross-validation on each of the four datasets. In this way, a dataset of size *n* will be partitioned into 10 folds each of size n/10. Nine folds are used for training a classification model, which is then evaluated on the rest one-fold data. The training and evaluation processes are repeated 10 times and a mean score is taken for precision, recall, and F-Measure.

Baseline Approaches: We consider the following two baselines approaches:

- Our first baseline is the approach proposed by Pal and Counts [33] as the baseline approach. Their approach uses only content features. They employ the Gaussian mixture model to cluster Twitter users into two groups and then pick one of the two groups as experts. They also rank Twitter users in this group based on their likelihood to be an expert. The Python package Gaussian Mixture[14] [34] is used for clustering in our experiments. We consider the following settings with respect to this baseline approach:
  - (PC^{Ev}): In this setting, we run Pal and Counts's approach to cluster all users in the evaluation data (the test data in our supervised approach) by ignoring the training data in the clustering process.
  - (PC^{Tr+Ev}): In this setting, we run Pal and Counts's approach to cluster all users in the training and evaluation data (basically the complete dataset used in our supervised approach).
- Our second baseline is based on Klout.[15] Klout is a system that calculates the influence score of social accounts across multiple social networks [38]. It uses a hierarchical combination of various feature scores aggregated over multiple social networks to calculate an influence score of a user, known as *KloutScore*. Klout offers a web API[16] through which we can obtain the KloutScore of a given Twitter user for a specific topic or domain. The score calculation is based on the approach outlined in [38, 47] and is an estimate of the percentile rank of a user's expertise for a given topic or domain. In this work, we consider any user with a *KloutScore* greater than 0.99 as an expert for that domain. These are users rated as those among the top 1% Twitter users with expertise in the domain. We refer to this baseline as KL.

6.3 Research Questions and Results

RQ1: *How effective is our specialized software guru recommendation approach?*

[14] http://scikit-learn.org/stable/modules/mixture.html.
[15] https://klout.com/home.
[16] https://klout.com/s/developers/research.

Table 6. Precision, Recall, and F-Measure of Our Approach

Domain	Precision	Recall	F-Measure
JavaScript	0.759	0.905	0.820
Android	0.655	0.755	0.681
Python	0.750	0.533	0.602
Linux	0.550	0.566	0.522
Average	0.678	0.690	0.656

Motivation and Approach: The more accurate a recommendation system is, the more beneficial it will be. To answer this research question, we investigate the effectiveness of our approach following the experiment setting described in Section 6.2.

Results: Table 6 shows the precision, recall, and F-Measure of our approach on four different domains. From Table 6, we observe that our approach can achieve an average F-Measure of 0.656 on the four domains. The average precision, recall, and F-Measure of our approach are 0.678, 0.690, and 0.656, respectively.

Also from Table 6, we can see that the F-Measure for the Linux domain is low, achieving a value of 0.522. To identify the reasons for a low F-Measure for Linux, we discussed with the labelers of our data and found that Linux experts are harder to identify than other experts. The reason is the people who are Linux experts share tweets across a wide range of topics, e.g., Linux kernel, Linux/Unix administration, Linux security, and so forth, and their scope is wider than those of other domains (e.g., JavaScript, etc.). This can be seen from the fact that the agreement among labelers, although still being fair, is lower for Linux than for other domains.

There have been many past studies that show results with F-Measure in the range of 0.5 to 0.7 [10, 40, 55, 56, 66]. The F-scores of our solution are also in this range. Higher F-Measures for domains such as JavaScript indicate better recommendation with less false positives and false negatives. Different users would have different tolerance for recommendation quality. Our results suggest that users would be happier when they use our approach for JavaScript than Linux. In any case, our results are better for all domains than those of baselines (as seen in RQ2).

RQ2: *Can our approach outperform existing Twitter expert recommendation approaches?*

Motivation and Approach: Our approach extends Pal and Counts's work [33] by proposing new features (i.e., nine network, eight profile features, and five GitHub features) and by using a two-stage classification process instead of a clustering technique. Since we extend this prior work, we need to demonstrate that our approach outperforms it. Also, we have compared our approach against Klout, which is a system that recommends users to follow given a particular topic or domain. To answer this research question, we follow the experimental settings described in Section 6.2 to compute the precision, recall, and F-Measure of Pal and Counts's approach [33] and Klout's approach [38]. We then compare and contrast their evaluation scores with those of ours.

Results: Table 7 shows the performance of the two variants of Pal and Counts's approach and the Klout baseline on the four different domains. From Table 7, we observe that our approach (shown in Table 6) can consistently achieve a better F-Measure than the baseline variants. In terms of F-Measure, which is a summary measure to evaluate if an increase in recall (precision) outweighs a reduction in precision (recall), our approach outperforms the Pal and Counts baseline variants for all domains by 7.63% to 166.10%. The Klout baseline is also outperformed by our approach on all domains by 18.79% to 31.47%.

Table 7. Precision, Recall, and F-Measure of the Baseline Approach Variants

Domain	Approach	Precision	Recall	F-Measure	Improvement
JavaScript	PC^{Ev}	0.740	0.465	0.563	45.61%
	PC^{Tr+Ev}	0.366	0.451	0.379	116.35%
	KL	0.898	0.561	0.690	18.79%
Android	PC^{Ev}	0.870	0.181	0.297	129.68%
	PC^{Tr+Ev}	0.202	0.354	0.256	166.10%
	KL	0.659	0.426	0.518	31.47%
Python	PC^{Ev}	0.933	0.278	0.423	42.33%
	PC^{Tr+Ev}	0.577	0.322	0.372	61.89%
	KL	0.808	0.356	0.494	21.86%
Linux	PC^{Ev}	0.925	0.332	0.485	7.63%
	PC^{Tr+Ev}	0.211	0.469	0.270	93.33%
	KL	0.500	0.339	0.404	29.21%

Improv. = Improvement in F-measure.

Table 8. Average F-Measure for Various Feature Combinations

Feature Setting	F-Measure	Performance Loss
ALL	0.656	-
ALL-GitHub	0.638	2.74%
ALL-Content	0.608	7.32%
ALL-Network	0.606	7.62%
ALL-Profile	0.451	31.25%
Only GitHub	0.180	72.56%
Only Content	0.167	74.54%
Only Network	0.271	58.69%
Only Profile	0.585	10.82%

RQ3: *What are important features that better differentiate specialized software gurus from nongurus?*

Motivation and Approach: In our approach, we use 32 different features to characterize a Twitter user, i.e., 10 content features, nine network features, eight profile features, and five GitHub features. In this research question, we want to evaluate the importance of each of the feature categories in predicting whether a Twitter user is a specialized software guru or not. To answer this research question, we take the dataset that we use to evaluate the performance of our approach in RQ1. We initially start with all the feature categories used in our dataset and ran experiments using our approach on various subsets of features. After that we removed one feature category at a time and repeated the experiments.

Results: Table 8 shows the various feature combinations that we have evaluated. The F-Measure scores shown in the table are averaged across all domains. Each row in the table corresponds to a set of features that is evaluated. The first row corresponds to the setting *ALL*, where we used all the features, namely, Content, Profile, Network, and GitHub features. Profile, Network, and GitHub features are the new categories of features that we propose in this work. Content features are the ones that were proposed by Pal and Counts [33]. Next, to measure the

strength of each category of features, we remove one category at a time and then calculate the corresponding F-Measures. The *ALL-GitHub* row refers to the setting where we use all features except those belonging to the GitHub category. Similarly, the rows *ALL-Content*, *ALL-Network*, and *ALL-Profile* refer to the settings where Content, Network, and Profile features were dropped and the remaining features evaluated. In order to evaluate the performance of using only a single category of features, we also add settings where features from only a single category are used for evaluation. The last four rows in Table 8 are related to it.

The results show that using a combination of all features achieves the maximum F-Measure of 0.656. Of the new categories of features we propose in this work, *Profile* features have the strongest predictive power. When we remove this feature, the F-Measure drops down by 31.25% to 0.451. Also, when we consider each feature category independently, the *Profile* features can achieve the highest F-Measure (i.e., 0.585), which shows its importance in predicting experts. The *Content* and *Network* features cause a drop of 7.32% and 7.62%, respectively, when removed. Individually, *Network* and *Content* features achieve an F-Measure of 0.271 and 0.167, respectively. *GitHub* features have a positive but very small contribution as removing them causes a drop of only 2.74%. Also, when we use GitHub features alone, only an F-Measure of 0.180 can be achieved.

The results above show that *Profile* features have the strongest discriminative power in discerning accounts of software gurus from others. As *Profile* features are based on text from external profile pages and Twitter bios of users, they contain words that can be used to identify experts. Also, in the *Profile* category, there are features that capture how long an account is present on Twitter and if it is a verified account. Such information is expected to strengthen the discriminative performance of Profile features and makes it perform better than other features. We also notice that *GitHub* features have the weakest performance as compared to all other feature categories. This is the case since in many cases developers do not share their *GitHub* profile links on Twitter, resulting in the value of *GitHub* features being zero.

RQ4: *Which feature values have the best predictive power across each domain?*

Motivation and Approach: In our approach, we use 32 different features to characterize a Twitter user, i.e., 10 content features, nine network features, eight profile features, and five GitHub features. In this research question, we want to evaluate the importance of each of the feature values in predicting whether a Twitter user is a specialized software guru or not. To answer this research question, we take the dataset that we use to evaluate the performance of our approach in RQ1. We use the procedure similar to what has been used in RQ3. We initially started with all the features used in our dataset and ran the experiments using our approach. After that we removed one feature at a time and repeated the experiments using our approach. For each domain, the F-Measure we obtained after removing each feature was compared to the domain's F-Measure obtained in Table 6 and the percentage drop was computed. The features that on removal cause the highest percentage drop in F-Measure are considered the most important. These top 10 features for each domain are shown in Table 9.

Results: In Table 9, for each domain, we report the top 10 features identified based on the percentage drop in F-Measure caused when the feature is removed. We also construct another list of important features based on the frequency in which they appear in the top 10 lists of the four domains. Table 10 shows the features that have appeared in the top 10 lists of at least two domains.

From Table 10, we can note that features across the four families, i.e., Network, Content, Profile, and GitHub, are important in differentiating specialized software gurus from others. The features PosBio, SS(Signal Strength), NExpertFollowers, GhRepos, and NegWeb are present across at least

Table 9. Top 10 Most Important Features for Each Domain

Rank	JavaScript	Android	Python	Linux
1	NegBio	PosBio	NCS	PosBio
2	CosSimWeb	PosWeb	NExpertsFollowed	NegBio
3	PageRank	NExpertsFollowed	IsGhMentioned	GhGists
4	AccountAge	SelfS	GhUserType	SS
5	PosBio	NCS	OutDegree Centrality	GhFollowers
6	GhRepos	CosSimTweetText	Degree Centrality	NExpertFollowers
7	LR	IsVerified	SS	CosSimTweetText
8	NExpertFollowers	NegWeb	NExpertFollowers	NegWeb
9	NegWeb	SS	NS	GhRepos
10	Friends	PageRank	GhRepos	Friends

Table 10. Most Important Features across the Four Domains

#Top 10 Lists	Feature Name	Dimension
3	PosBio	Profile
3	SS	Content
3	NExpertFollowers	Network
3	GhRepos	GitHub
3	NegWeb	Profile
2	NegBio	Profile
2	NExpertsFollowed	Network
2	NCS	Content
2	CosSimTweetText	Profile
2	PageRank	Network
2	Friends	Network

three domains. However, only the Profile feature PosBio is present in the top 5 ranks across the three domains. In addition to Pos Bio and NegWeb, other important Profile features are NegBio and CosSimTweetText. Network features NExpertsFollowed and Friends are also present in the list for at least two domains.

From Table 9, it can be observed that the Profile features are the most frequent among the top 10 features and at relatively higher ranks. This is in line with the results observed in Table 8, where removing the Profile category had caused the highest drop in F-Measure. The probabilities extracted from users' webpage and biodata seem to have more discriminative power as compared to other features. Among Network features, NExpertFollowers has the strongest impact. This makes sense as a user who is followed by other experts is expected to have a high probability of being an expert.

RQ5: *What is the cross-domain performance of our approach?*

Motivation and Approach: There are many other software engineering domains aside from the four considered in this work. Thus, we need to check if a model learned from one domain can possibly be used to identify experts from another domain. To answer this research question, we perform experiments in which we train our model based on training data from one domain and then use this model to identify gurus in other domains.

Table 11. F-Measure of our Approach When Evaluated on Cross-Domain Setting

Test Domain	Setting	Train Domain	F-Measure
JavaScript	cross-domain	Android	0.777
		Python	0.784
		Linux	0.742
		Average	0.768
	within-domain	JavaScript	0.820
Android	cross-domain	JavaScript	0.658
		Python	0.615
		Linux	0.588
		Average	0.620
	within-domain	Android	0.681
Python	cross-domain	JavaScript	0.616
		Android	0.604
		Linux	0.479
		Average	0.566
	within-domain	Python	0.602
Linux	cross-domain	JavaScript	0.485
		Android	0.388
		Python	0.466
		Average	0.446
	within-domain	Linux	0.522
Average	cross-domain	-	0.600
	within-domain	-	0.656

Results: Table 11 shows the performance of our model when trained on each domain and tested on each of the other three domains. We refer to this setting as the *cross-domain* setting. On average, we are able to achieve an F-Measure of 0.600 in the *cross-domain* setting. Note that our approach was able to achieve an average F-Measure of 0.656 when the test and train data is from the same domain—see Table 6 (we refer to as *within-domain* setting). Thus, there is only a small drop in F-Measure (i.e., 0.056), which shows that our approach is effective for the *cross-domain* setting. Labeled data from one domain can be used to build an effective model to predict experts from other domains with only a small penalty in performance. In order to check if the F-Measure obtained in the *cross-domain* result is significantly different from the F-Measure obtained in the *within-domain* setting, we performed the Mann-Whitney U test [29] on the means of F-Measures obtained in the *cross-domain* setting and *within-domain* setting. The test gave a p-value of 0.055, which is greater than 0.05, based on which we can say that there is no statistical difference between the *within-domain* and *cross-domain* results.

For the *cross-domain* setting, it can be observed that the performance of our approach for domain Linux when it is trained using data from domain Android is quite low, despite both being operating systems. To understand the reason behind this observation, we compare the contents of tweets in our Linux and Android datasets. We find that the vocabulary used by Linux experts is rather different than that used by Android experts. Most Android tweets are at the application level (e.g., how to validate Android in-app subscription purchase), while Linux tweets are at the system level (e.g., how to enable AES-NI advanced encryption on the Linux system).

6.4 Threats to Validity

Threats to internal validity relate to errors in our experiments and our labeling. We have checked our code multiple times; still there could have been errors that we did not notice. At times it is hard for our user study participants to decide whether someone is an experienced domain-specific practitioner or whether a set of tweets is helpful for others or not. To deal with such cases, we allow participants to choose the "Can't Determine" option and omit those cases from our dataset to improve the quality of the ground-truth labels. We also measure the agreement rate among the participants. To do this, we have computed interrater agreement for the labeling task using the measure of Cohen's Kappa [11]. As can be seen from Table 4, except for the Linux dataset, the agreement is at least moderate. For the Linux dataset, the agreement is still fair. These results show that the raters in general agree with one another; thus, the threat introduced due to disagreement among raters is minimal. Our strategy of omitting "Can't Determine" cases may bias the evaluation to easier cases. To investigate this threat, we have relooked into these "Can't Determine" cases. We found that many of these cases are less interesting ones; e.g., such users often only post a few domain-related tweets, include little information in their profile, and so forth. They are less likely to be interesting domain experts to be followed.

Threats to external validity relate to the generalizability of our approach. To mitigate this threat, in this article, we have evaluated our approach on Twitter users belonging to two domain types, i.e., *programming languages* (which include JavaScript and Python domains) and *operating systems* (which include Android and Linux domains). We have also run experiments to check for cross-domain performance to evaluate the generalizability of our approach. In the future, to further reduce the threats to external validity, we plan to evaluate our approach on even more domains and domain types.

The generalizability of our results may also be impacted by the use of GitHub features. Some developers may not be using GitHub and for them we will not have their GitHub features. For such cases the performance of our approach may be slightly lower, as removing the GitHub features causes a drop of about 2.74% (see Table 8). It is possible to extract similar features from other coding websites such as BitBucket, which we leave as future work. We focus on GitHub in our work as it is currently the most popular social coding platform and is also growing fast.[17] In our dataset of Twitter users used for experiments, 18.89% (116/614) of them have GitHub links in their profiles. On the other hand, only 1.14% (7/614) of users have BitBucket links in their profiles. Note that the collection of users in our dataset is not biased in any way toward GitHub.[18] Many past studies have also focused on GitHub due to its popularity [41, 45, 48, 49, 57, 58].

Another factor that may impact the generalizability of our results may be the use of threshold values that are used to determine domain experts. As mentioned in Section 6.1, in this work we identify gurus among users who post more than 10 domain-related tweets. To check the impact of this number, we evaluated the performance of our expert identification approach among users who post more than 20 or 30 domain-related tweets. We find that there is only a small change in the F-Measure (an increase in F-Measure by 0.5% to 4.3% when we increase the threshold to a higher number) provided that the remaining number of data points left after filtering at higher threshold levels is at least 50. If we have few data points left after filtering, then the classifier is not able to learn a good model—which is as expected. Also, by default, for Klout we use a KloutScore threshold of 0.99. We checked for change in its performance if the threshold is decreased below

[17] https://octoverse.github.com/.

[18] The users in our dataset are collected by initially merging several seed lists of popular software developers present on Twitter. The resultant combined set is then expanded to include users who are followed or follow a certain number of users in the combined set; c.f. Section 5.

Table 12. Feature Strength versus Classification Approach

Classification Approach	Features	
	All Features	Content Only
2-Stage Classification	0.656	0.167
PC^{Ev}	0.497	0.452
PC^{Tr+Ev}	0.403	0.304

0.99. We found little change in performance (a decrease in F-Measure by 0% to 1.21%) when we vary the threshold from 0.75 to 0.99 (using a step of 0.03).

Threats to construct validity relate to the suitability of our evaluation metric. In this article, we use precision, recall, and F-Measure. These metrics are well known and have been used in many past studies, e.g., [23, 63, 67]. To further investigate the effectiveness of our proposed approach, we perform a user study among some Android developers, to check if they accept the recommendations generated by our approach and the baselines. The details of the study are discussed in Section 7.2. The study finds that the recommendations provided by our approach are at least 25.93% more accurate than the baselines.

7 DISCUSSION

7.1 Benefits of Adding New Features and Employing Our New Classification Method

In our work, we have proposed three new categories of features as well as a two-stage classification approach. Here, we perform experiments to evaluate the individual contribution of the set of new features and the new classification approach in achieving better performance over baselines. Specifically, we check the performance of our two-stage classification approach on only the Content features, and also the performance of the baseline approach on all features combined.

Table 12 shows the results of our experiments. From Table 12, we observe that a combination of our two-stage classification approach and all the features can achieve an F-Measure of 0.656. However, when we run our two-stage classification approach on only Content features, the F-Measure drops down to 0.167. This shows that without the new category of features our two-stage classification approach is not able to achieve good performance. Next, we evaluate the performance of two variants of the Pal and Counts [33] approach using features from all categories. Table 12 shows that the F-Measure drops down to 0.497 for the PC^{Ev} baseline variant and 0.403 for the PC^{Tr+Ev} variant. This shows that our two-stage classification approach is able to achieve better performance over the baseline approach of Pal and Counts [33] when all the features are used.

7.2 Do Developers Follow Recommendations Provided by Our Approach?

We conduct a user study to compare the performance of our approach with the baseline approaches. For this purpose, we use a dataset of Twitter users belonging to the "Android" domain that we collected earlier—see Table 5. We divide this dataset into two approximately equal-sized subsets. We randomly choose one of them for training and the other one to generate recommendations from. We ran our proposed approach and the baselines, i.e., Klout, PC^{Ev}, and PC^{Tr+Ev}, on this dataset.

After we have the results from all four approaches, we chose the top 3 users returned by each approach and randomly mixed them together and removed duplicates. These users were then shown to some Android developers. For each user, the user's Twitter profile as well as latest tweets were shown to the developers. A single question was asked about each user to each developer: "Are you interested in following the above Twitter account, so that following it may help you in getting

Table 13. Converting Answers to Ratings

Answer Chosen	Rating
NO, I am not interested to follow the account shown above	0
I already follow the account shown above	1
YES, I am interested to follow the account shown above	1

Table 14. User Study Results

Approach	NDCG@3
KL	0.43
PC^{Ev}	0.54
PC^{Tr+Ev}	0.54
Our	0.68

updated information related to Android programming?" As an answer to this question, each developer was asked to choose one out of the following three options: "(A) NO, I am not interested to follow the account shown above," "(B) YES, I am interested to follow the account shown above," and "(C) I already follow the account shown above." To get Android developers as participants of our user study, we randomly browsed for relevant accounts on Twitter. Next, the first author of the article personally contacted each of them through Twitter messaging service. This process is similar to the process used for contacting developers for the initial survey as described in Section 3. We managed to attract 10 developers who agreed to participate in our user study.

The answers provided by each user were converted into binary ratings following the conversion table shown in Table 13. To evaluate the results of our user study, we make use of Normalized Discounted Cumulative Gain (NDCG) [19]. NDCG is commonly used to measure the performance of information retrieval and recommendation systems [25]. The value of the NDCG metric varies from 0 to 1, with 1 representing the ideal ordering. The following equation is used to compute NDCG, where rel_i is the rating assessment provided by a user at position i in the ranking:

$$NDCG@3 = \frac{1}{IDCG} \sum_{i=1}^{3} \frac{rel_i}{log(i+1)}.$$

Table 14 shows the results of our user study for each of the four approaches (ours and the three baselines). We can see from Table 14 that the NDCG score of our proposed approach is 0.68, which is the highest when compared to baselines. In terms of NDCG, our approach outperforms Klout, PC^{Tr+Ev}, and PC^{Ev} by 58.14%, 25.93%, and 25.93%, respectively. The results of the user study further highlight the effectiveness of our proposed approach in recommending domain experts.

7.3 Lessons Learned

We share some points below that may be helpful to researchers interested in exploring problems similar to what has been done in this work:

- *Design effective and comprehensive features*: Every dataset, platform, and problem is different. To recommend experts on Twitter, we designed a comprehensive set of features by analyzing the nature of the problem and data that we have. This results in the construction of a more effective recommendation system. Based on this experience, we recommend

future studies, especially those that build recommendation systems for a new dataset or problem, to look into unique characteristics of the data and problem. A good understanding of these characteristics is needed to create new features that would be instrumental in the construction of effective recommendation systems. Having effective features can be more important than deploying more powerful machine-learning algorithms in terms of their impact on recommendation quality.

- *Incorporate external resources*: In our study, we find that features extracted from linked external resources such as personal web pages of users and GitHub profiles help in improving recommendation quality. Thus, researchers interested in solving similar problems may want to go beyond data coming from one source. Linked external resources can provide additional insights into the problem at hand.
- *Disseminate survey strategically*: Researchers often email their surveys to developers present on GitHub [45, 49, 65]. While this method may work well and can result in response rates greater than 15%, sometimes in cases where the study focus is a specific software engineering community, the response rates may go down. This is the case of a recent study focusing on Slack [24], where the response rate was 7.84% (51/650). In our initial survey, we received a poor response rate of less than 10% when we contacted GitHub developers randomly sampled from GHTorrent [15]. This may have happened as the sample chosen from GitHub may not have been representative of developers who use Twitter. Based on this outcome, we started a brand new survey and contacted developers who were actually present on Twitter, using personalized Twitter messages. The procedure is described in Section 3. This time the response rate improved to 17.84%. Thus, one of the takeaways from our work is that if the research problem being addressed caters to a specific software engineering community, sampling should be done from a population of that specific community only. Additionally, instead of mass-mailing developers, personally contacting developers using channels often used by members of the target community, e.g., Twitter direct messaging in our case, also helps in achieving more responses.

8 CONCLUSION AND FUTURE WORK

Twitter is becoming increasingly popular and has changed the way people share information and collaborate with one another. Singer et al. report that software developers use Twitter to get awareness of people and trends, extend their technical knowledge, and build connections with other developers. They also report that it is challenging for developers to find interesting users to follow [45]. To better understand developers' needs, we first conduct an online survey with 38 developers. For those who use Twitter in their software development activities, we ask about the kinds of users they would like to follow to help in their software development activities. The results of our survey show that most developers would like to follow specialized software gurus, e.g., experts in Python. Based on the survey result, we propose a new approach that can automatically recommend software gurus of a specialized domain (e.g., Python).

Our approach makes use of 32 features from four dimensions (i.e., Content, Network, Profile, and GitHub) to characterize a Twitter user. It then uses a two-stage classification technique that analyzes a set of labeled training data to create a discriminative model that can differentiate specialized software gurus from other domain-related Twitter users. In our experiment, we have evaluated our approach to classify domain-related Twitter users from four domains, i.e., JavaScript, Android, Python, and Linux, into two categories (specialized gurus and others). The experiment results show that our approach can achieve F-Measure scores of 0.522 to 0.820 on the four domains. Our approach can improve the F-Measures achieved by baseline approaches [33, 38] by at least 7.63%.

As a future work, we plan to consider more tweets and domain-related users and evaluate our model on more domains in addition to the four considered in this work. We also plan to build and deploy a live system (e.g., as a website or an Android app) that can continuously extract data from Twitter and recommend domain-specific experts and promote this system to developers. We also plan to do studies to understand what kind of Twitter accounts developers tend to unfollow after following them for some time. Understanding characteristics of such accounts can help us build a system to recommend potential accounts to unfollow and thus better help developers in carefully curating the list of accounts they follow.

REFERENCES

[1] Palakorn Achananuparp, Ibrahim Nelman Lubis, Yuan Tian, David Lo, and Ee-Peng Lim. 2012. Observatory of trends in software related microblogs. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 334–337.

[2] Maurício Aniche, Christoph Treude, Igor Steinmacher, Igor Wiese, Gustavo Pinto, Margaret-Anne Storey, and Marco Aurélio Gerosa. 2018. How modern news aggregators help development communities shape and share knowledge. In *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)*. ACM, New York, NY, USA, 499–510.

[3] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*. ACM, 361–370.

[4] John Anvik and Gail C. Murphy. 2007. Determining implementation expertise from bug reports. In *Proceedings of the 4th International Workshop on Mining Software Repositories*. IEEE Computer Society, 2.

[5] Gargi Bougie, Jamie Starke, Margaret-Anne Storey, and Daniel M. German. 2011. Towards understanding Twitter use in software engineering: Preliminary findings, ongoing challenges and future questions. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*. 31–36.

[6] Ulrik Brandes. 2008. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks* 30, 2 (2008), 136–145.

[7] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: Experiences from the Scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

[8] John L. Campbell, Charles Quincy, Jordan Osserman, and Ove K. Pedersen. 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological Methods & Research* 42, 3 (2013), 294–320.

[9] Shuo Chang and Aditya Pal. 2013. Routing questions for collaborative answering in community question answering. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM, 494–501.

[10] Morakot Choetkiertikul, Daniel Avery, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. 2015. Who will answer my question on stack overflow? In *2015 24th Australasian Software Engineering Conference (ASWEC'15)*. IEEE, 155–164.

[11] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.

[12] Juliet Corbin, Anselm Strauss, and Anselm L Strauss. 2014. *Basics of Qualitative Research*. Sage.

[13] Linton C. Freeman. 1979. Centrality in social networks conceptual clarification. *Social Networks* 1, 3 (1979), 215–239.

[14] GitHub. 2015. About GitHub Inc. Retrieved from https://github.com/about/press. Accessed August 27, 2015.

[15] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*. IEEE Press, Piscataway, NJ, 233–236.

[16] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. 2016. A needle in a haystack: What do Twitter users say about software?. In *2016 IEEE 24th International Requirements Engineering Conference (RE'16)*. IEEE, 96–105.

[17] Chaoran Huang, Lina Yao, Xianzhi Wang, Boualem Benatallah, and Quan Z. Sheng. 2017. Expert as a service: Software expert recommendation via knowledge domain embeddings in stack overflow. In *2017 IEEE International Conference on Web Services (ICWS'17)*. IEEE, 317–324.

[18] William Hudson. 2013. Card sorting. In *The Encyclopedia of Human-Computer Interaction*. 2nd ed.

[19] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[20] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 96–107.

[21] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2017. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering* 1 (2017), 1–1.

[22] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue B. Moon. 2010. What is Twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*. 591–600.

[23] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 308–311.

[24] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 2016. Why developers are slacking off: Understanding how software teams use slack. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*. ACM, 333–336.

[25] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval*. Springer Science & Business Media.

[26] David Lo, Nachiappan Nagappan, and Thomas Zimmermann. 2015. How practitioners perceive the relevance of software engineering research. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 415–425.

[27] David Ma, David Schuler, Thomas Zimmermann, and Jonathan Sillito. 2009. Expert recommendation with usage expertise. In *IEEE International Conference on Software Maintenance, 2009 (ICSM'09)*. IEEE, 535–538.

[28] Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. 2015. Code, camera, action: How software developers document and share program knowledge using YouTube. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*. IEEE Press, 104–114.

[29] Henry B. Mann and Donald R. Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics* 18 (1947), 50–60.

[30] Christopher D. Manning, Prabhakar Raghavan, and Hinrich SchÃijtze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[31] Mary L. McHugh. 2012. Interrater reliability: The kappa statistic. *Biochemia Medica* 22, 3 (2012), 276–282.

[32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web*. Technical Report 1999-66. Stanford InfoLab.

[33] Aditya Pal and Scott Counts. 2011. Identifying topical authorities in microblogs. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*. ACM, 45–54.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (Oct. 2011), 2825–2830.

[35] Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, Rocco Oliveto, Mir Hasan, Barbara Russo, Sonia Haiduc, and Michele Lanza. 2016. Too long; didn't watch!: Extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 261–272.

[36] Martin F. Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.

[37] Philips Kokoh Prasetyo, David Lo, Palakorn Achananuparp, Yuan Tian, and Ee-Peng Lim. 2012. Automatic classification of software related microblogs. In *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM'12)*. IEEE, 596–599.

[38] Adithya Rao, Nemanja Spasojevic, Zhisheng Li, and Trevor DSouza. 2015. Klout score: Measuring influence across multiple social networks. In *2015 IEEE International Conference on Big Data (Big Data'15)*. IEEE, 2282–2289.

[39] Johnny Saldaña. 2015. *The Coding Manual for Qualitative Researchers*. Sage.

[40] Gustavo Santos, Klérisson V. R. Paixão, Nicolas Anquetil, Anne Etien, Marcelo de Almeida Maia, and Stéphane Ducasse. 2017. Recommending source code locations for system specific transformations. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*. IEEE, 160–170.

[41] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. 2017. Cataloging Github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 314–319.

[42] Abhishek Sharma, Yuan Tian, and David Lo. 2015. NIRMAL: Automatic identification of software relevant tweets leveraging language model. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER'15)*. IEEE, 449–458.

[43] Abhishek Sharma, Yuan Tian, and David Lo. 2015. What's hot in software engineering Twitter space? In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME'15)*. IEEE, 541–545.

[44] Abhishek Sharma, Yuan Tian, Agus Sulistya, David Lo, and Aiko Fallas Yamashita. 2017. Harnessing Twitter to support serendipitous learning of developers. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*. IEEE, 387–391.

[45] Leif Singer, Fernando Marques Figueira Filho, and Margaret-Anne D. Storey. 2014. Software engineering at the speed of light: How developers stay current using Twitter. In *36th International Conference on Software Engineering (ICSE'14)*. 211–221.

[46] Edward K. Smith, Christian Bird, and Thomas Zimmermann. 2015. Build it yourself!: Homegrown tools in a large software company. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*. IEEE Press, 369–379.
[47] Nemanja Spasojevic, Jinyun Yan, Adithya Rao, and Prantik Bhattacharyya. 2014. Lasta: Large scale topic assignment on multiple social networks. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1809–1818.
[48] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. 2014. The (r) evolution of social media in software engineering. In *Proceedings of the Future of Software Engineering*. ACM, 100–116.
[49] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. 2017. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering* 43, 2 (2017), 185–204.
[50] Anselm Strauss and Juliet M. Corbin. 1997. *Grounded Theory in Practice*. Sage.
[51] Yuan Tian, Palakorn Achananuparp, Ibrahim Nelman Lubis, David Lo, and Ee-Peng Lim. 2012. What does software engineering community microblog about? In *MSR*. 247–250.
[52] Yuan Tian and David Lo. 2014. An exploratory study on software microblogger behaviors. In *MUD*.
[53] Yuan Tian, David Lo, and Julia Lawall. 2014. Automated construction of a software-specific word similarity database. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE'14)*. IEEE, 44–53.
[54] Twitter. 2017. About Twitter Inc. Retrieved from https://about.twitter.com/company. Accessed July 26, 2017.
[55] Gias Uddin and Foutse Khomh. 2017. Automatic summarization of API reviews. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE'17)*. IEEE, 159–170.
[56] Harold Valdivia Garcia and Emad Shihab. 2014. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 72–81.
[57] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. 2016. The sky is not the limit: Multitasking on GitHub projects. In *International Conference on Software Engineering (ICSE'16)*. ACM, 994–1005. Retrieved from DOI: https://doi.org/10.1145/2884781.2884875.
[58] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. StackOverflow and GitHub: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing (SocialCom'13)*. IEEE, 188–195.
[59] Anthony J. Viera and Joanne M. Garrett. 2005. Understanding interobserver agreement: The kappa statistic. *Family Medicine* 37, 5 (2005), 360–363.
[60] Xiaofeng Wang, I. Kuzmickaja, K.-J. Stol, P. Abrahamsson, and B. Fitzgerald. 2014. Microblogging in open source software development: The case of Drupal and Twitter. *IEEE Software* 31, 4 (2014), 72–80.
[61] Scott White and Padhraic Smyth. 2003. Algorithms for estimating relative importance in networks. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 266–275.
[62] Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering* 43, 3 (2017), 272–297.
[63] Xin Xia, David Lo, Emad Shihab, Xinyu Wang, and Xiaohu Yang. 2015. ELBlocker: Predicting blocking bugs with ensemble imbalance learning. *Information and Software Technology* 61 (2015), 93–106.
[64] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. 2015. Dual analysis for recommending developers to resolve bugs. *Journal of Software: Evolution and Process* 27, 3 (2015), 195–220.
[65] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. 2017. The impact of continuous integration on other software development practices: A large-scale empirical study. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 60–71.
[66] Pingyi Zhou, Jin Liu, Zijiang Yang, and Guangyou Zhou. 2017. Scalable tag recommendation for software information sites. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*. IEEE, 272–282.
[67] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. 2014. Combining text mining and data mining for bug report classification. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*. IEEE, 311–320.

Received June 2016; revised March 2018; accepted July 2018