SOCIAL SOFTWARE DEVELOPMENT: INSIGHTS AND SOLUTIONS

ABHISHEK SHARMA

SINGAPORE MANAGEMENT UNIVERSITY

2018

Social Software Development: Insights and Solutions

by

Abhishek SHARMA

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Information Systems

Singapore Management University
2018

Social Software Development: Insights and Solutions

Abhishek SHARMA

Abstract

Over last few decades, the way software is developed has changed drastically. From being an activity performed by developers working individually to develop standalone programs, it has transformed into a highly collaborative and cooperative activity. Software development today can be considered as a participatory culture, where developers coordinate and engage together to develop software while continuously learning from one another and creating knowledge.

In order to support their communication and collaboration needs, software developers often use a variety of social media channels. These channels help software developers to connect with like-minded developers and explore collaborations on software projects of interest. However, developers face a lot of challenges while trying to make use of various social media channels. As the volume of content produced on social media is huge developers often face the problem of information overload while using these channels. Also creating and maintaining a relevant network among a huge number of possible connections is challenging for developers. The works performed in this dissertation focus on addressing the above challenges with respect to Twitter, a social media popular among developers to get the latest technology updates, as well as connect with other developers. The first three works performed as a part of this dissertation deal with understanding the software engineering content produced on Twitter and how it can be harnessed for automatic mining of software engineering related knowledge. The last work aims at understanding what kind of accounts software developers follow on Twitter, and then proposes an approach which can help developers to find software experts on Twitter. The following paragraphs briefly describe the works that have been completed as part of this dissertation and how they address the aforementioned challenges.

In the first work performed as part of the dissertation, an exploratory study was conducted to understand what kind of software engineering content is popular among developers in Twitter. The insights found in this work help to understand the content that is preferred by developers on Twitter and can guide future techniques or tools which aim to extract information or knowledge from software engineering content produced on Twitter. In the second work, a technique was developed which can automatically differentiate content related to software development on Twitter from other non-software content. This technique can help in creating a repository of software related content extracted from Twitter, that can be used to create downstream tools which can do tasks such as mining opinions about APIs, best practices, recommending relevant links to read, etc. In the third work, Twitter was leveraged to automatically find URLs related to a particular domain, as Twitter makes it possible to infer the network and popularity information of users who tweet a particular URL. 14 features were proposed to characterize each URL by considering webpage contents pointed by it, popularity and content of tweets mentioning it, and the popularity of users who shared the URL on Twitter.

In the final work of this dissertation, an approach has been proposed to address the challenge developers face in finding relevant developers to follow on Twitter. A survey was done with developers, and based on its analysis, an approach was proposed to identify software experts on Twitter, provided a given software engineering domain. The approach works by extracting 32 features related to Twitter users, with features belonging to the categories such as Content, Network, Profile, and GitHub. These features are then used to build a classifier which can identify a Twitter user as a software expert of a given domain or otherwise. The results show that our approach is able to achieve F-Measure scores of 0.522-0.820 on the task of identifying software experts, achieving an improvement of at-least 7.63% over the baselines.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. David Lo, for accepting me as a graduate student and consistently supporting me throughout my PhD journey. I greatly appreciate the way he patiently taught me how to approach research problems, handle impediments, and encouraged me to develop independent thinking. I admire his ability to motivate people towards completing goals with a positive and enthusiastic attitude, and am thankful for the guidance provided by him for my dissertation research.

I am grateful to the other members of my dissertation committee, Dr. Jing Jiang and Dr. Hady Wirawan Lauw for providing valuable suggestions and comments, which gave me exposure to a different perspective on approaching research problems. I would also like to thank Dr. Nachiappan Nagappan for taking time to review my dissertation and providing me valuable feedback.

I would like to thank my friends and colleagues in SIS Research Center: Doyen Sahoo, James Arambam, Maksim Tkachenko, Kenny Choo, Camellia Zakaria, and Ferdian Thung, who supported me both intellectually and personally. Also. I am grateful for the great learning experience I had working with the members of the SOAR (SOftware Analytics Research) group.

I am also thankful to Singapore Management University for providing me with generous scholarship and healthy environment for research. The support provided by the members of administration staff Seow Pei Huan, Yeo Yar Ling and Ong Chew Hong is much appreciated. Also I would like to thank LARC (Living Analytics Research Centre) for providing me access to their computing infrastructure to perform experiments as well as supporting me for a conference.

Last but not the least I will be always indebted to my family, father Dr. Sandeep Kumar Sharma and mother Sunila Bhardwaj, who always believed in me when even I did not, my wife Anamika who motivates me every day to do better, and sister Apurva who looks up to me for guidance, pushing me to learn more.

Chapter 1

Introduction

This chapter discusses the motivation of the problems which are addressed in this dissertation. The chapter provides a summary of works completed, and the structure of this dissertation proposal.

1.1 Motivation

The increased adaptation of Internet and widespread rise of various social media channels over past decade has revolutionized the way people access and share information. The social media revolution has also changed the way software developers communicate, collaborate, and learn. The availability of various social media channels and tools has given rise to the emergence of *social programmer* [109, 93], who actively engages with social media channels in order to learn new things, communicate ideas, and collaborate with other programmers to develop software.

Software developers make use of a variety of social media channels. These include domain specific channels such as Stack Overflow[1], GitHub[2], etc., as well as domain agnostic channels such as Twitter[3], Reddit[4], HackerNews[5], etc. Storey et

[1]`https://stackoverflow.com/`
[2]`https://github.com/`
[3]`https://twitter.com/`
[4]`https://www.reddit.com`
[5]`https://news.ycombinator.com/`

al. have examined in detail the role social media channels play in software development [96, 94]. They found that software developers use social media channels for supporting various activities such as staying up-to-date, find answers to technical questions, coordinate with other developers on projects, learn new skills, discover and connect with other developers, display skills and accomplishments etc. By supporting such activities various social media channels promote a participatory culture of software development, where the software is co-developed by developers spread across the globe, who continuously interact with each other through social media channels.

As discussed above developers derive a lot of benefits from various social media channels in order to support their software development activities. However, using such social media channels has its own set of challenges. As explored in [96] some of these challenges are information overload, constant distraction, collaboration and participation hurdles, etc. In this dissertation, techniques have been proposed to address some of these challenges with respect to *Twitter*, a microblog based social medium, which is also popular among software development community [96]. The insights found and techniques proposed in this dissertation in order to solve challenges software developers face in using Twitter, would also be applicable in addressing similar problems in other social media used by developers. In next paragraph, a description is provided of how developers use Twitter and the challenges faced by them whose possible solutions have been proposed in this dissertation.

Microblogs such as *Twitter* are the fifth most popular social media used by developers [96]. A recent study by Singer et al. found that software developers use Twitter to “keep up with the fast-paced development landscape” [89]. Unfortunately, due to the general purpose nature of Twitter, it’s challenging for developers to use Twitter for their development activities. Singer et al. found that *consuming content* and *maintaining a relevant network* are the two main challenges developers face while using Twitter for software development [89]. Twitter being a platform open to domains other than software development, has content which is large in variety as well as volume. This results in the problem of *information overload* for developers who use Twitter for gaining knowledge related to software development only. Further, Twitter supports a wide variety of user accounts such as personal accounts, organizational accounts, bot accounts etc. Also, the number of accounts present on Twitter is very large. Due to these reasons, it becomes very challenging for developers to decide which *accounts to follow* and which not to. In this dissertation, some approaches have been proposed which help in mitigating these two challenges. In the following sections, an overview is given of the works completed as part of this thesis.

1.2 Contribution Summary

As discussed in previous section, Singer et al. had found that the two main challenges faced by software developers are *information overload* and deciding which *accounts to follow* [89]. In this dissertation, we propose four works which try to address the two aforementioned challenges. The first three works aim at solving the problem of *information overload*. In the first work, we did an exploratory study to identify what are the popular events related to software engineering on Twitter. In the second work, we propose an approach that can identify software relevant tweets on Twitter. In the third work, we propose a set of features that can be used to characterize web links related to software engineering shared on Twitter. These features were then evaluated for their classification performance using supervised and unsupervised methods. In the final work, we propose a technique that can help developers to find expert users relevant to a given software engineering domain, which they can then follow on Twitter. We give a brief summary of each of the completed works below.

Understanding Popular Software Engineering Content Produced in Social Networks

In this work, we performed an exploratory study on software engineering related events in Twitter. A large set of Twitter messages was collected over a period of 8 months that were made by 90,883 Twitter users and then filtered on five programming language keywords. To the best of our knowledge, no previous work in software engineering domain has analyzed such a huge amount of content related to software engineering tweets. A state-of-the-art Twitter event detection algorithm [24] borrowed from the Natural Language Processing (NLP) domain was then run on the data. Next, using the open coding procedure, 1,000 events that are identified by the NLP tool were manually analyzed, and eleven categories of events (10 main categories + "others") created. It was found that external resource sharing, technical discussion, and software product updates are the "hottest" categories. These findings shed light on hot topics in Twitter that are interesting to many people and they provide guidance to future Twitter analytics studies that develop automated solutions to help users find fresh, relevant, and interesting pieces of information from Twitter stream to keep developers up-to-date with recent trends.

Automatic Identification of Software Relevant Content in Social Media

In this work, to help developers cope with noise, we propose a novel approach named NIRMAL, which automatically identifies software relevant tweets from a collection or stream of tweets. The approach is based on language modeling which learns a statistical model based on a training corpus (i.e., set of documents). A subset of posts from Stack Overflow, a programming question and answer site, was used as a training corpus to learn a language model. A corpus of tweets was then used to test the effectiveness of the trained language model. The tweets were sorted based on the rank the model assigned to each of the individual tweets. The top 200 tweets were then manually analyzed to verify whether they are software related or not, and then an accuracy score was calculated. The results of our experiments show that NIRMAL can achieve accuracy@K scores of up to 0.900, beating a random baseline by 192%. Also when NIRMAL is combined with a keyword based approach, it improves the performance by up to 31% of the keyword only approach.

Mining Informative Online Resources Shared by Developers on Social Media

Developers often rely on various online resources, such as blogs, to keep themselves up-to-date with the fast pace at which software technologies are evolving. Singer et al. found that developers tend to use channels such as Twitter to keep themselves updated and support learning, often in an undirected or serendipitous way, coming across things that they may not apply presently, but which should be helpful in supporting their developer activities in future. However, identifying relevant and useful articles among the millions of pieces of information shared on Twitter is a non-trivial task. In this work to support discovery of relevant and informative resources to support developer learning, an unsupervised and a supervised approach was proposed to find and rank URLs (which point to web resources) harvested from Twitter based on their informativeness and relevance to a domain of interest. 14 features were proposed to characterize each URL by considering contents of webpage pointed by it, contents and popularity of tweets mentioning it, and the popularity of users who shared the URL on Twitter. To evaluate the performance of the proposed methods we make use of Normalized Discounted Cumulative Gain (NDCG) [39]. The value of the NDCG metric varies from 0 to 1, with 1 representing the ideal ordering. The results of our experiments show that the proposed unsupervised and supervised approaches can achieve NDCG scores of 0.719 and 0.832 respectively. As these scores are not very far off from 1, the approaches do show promise in showing the relevant articles at higher ranks.

Recommending Experts in the Software Engineering Twitter Space

As highlighted by Singer et al. in [89], a common challenge faced by developers on social networks such as Twitter is to *maintain a relevant network*. Developers often connect with new people and disconnect with accounts which they think are not helping them much in supporting their software development related activities.

In this work, to help developers in finding relevant people to follow and connect with, we propose an approach to identify specialized software gurus. To understand the type of accounts developers like to follow, we conducted a survey with 36 developers who use Twitter in their development activities. The results of the survey highlighted that developers are interested in following specialized software experts who share relevant technical tweets. Based on the survey results, an approach has been designed which first extracts different kinds of features that characterize a Twitter user. It 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). The effectiveness of the proposed approach in finding specialized software gurus was evaluated for four different domains (JavaScript, Android, Python, and Linux). The results show that proposed approach can differentiate specialized software experts from other domain-related Twitter users with an F-measure of up to 0.820, beating baseline approaches by at least 7.63%.

1.3 Structure of this Dissertation

The outlines of next chapters are described here. In Chapter 2 a review of some previous work related to social media in software engineering present in literature has been performed. Chapter 3 describes a study in which we explored the popular content related to software engineering in Twitter. Chapter 4 describes NIRMAL, a technique which helps to automatically identify tweets related to software engineering on Twitter. In Chapter 5 we describe an approach to mine informative URLs and links related to software engineering shared on Twitter. Chapter 6 presents an approach to recommend experts on Twitter related to a specialized software domain (e.g., Python). Chapter 7 concludes this dissertation and presents the presents some future directions.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, we discuss the work in literature related to software engineering and social media. The chapter has 2 sections. In Section 2.2 we describe studies which have explored how software developers use various social media to support software development. In Section 2.3 we discuss various tools, and/or techniques that have been proposed in the literature which extract software engineering knowledge from various social media and help software developers better utilize such channels.

2.2 Social Media and Software Development

In the past few years, a lot of attention has been given in research on how software developers use social media channels to aid their day to day work. Storey et al. in [95] advocated for increased research to understand the benefits, risks, and limitations of social media use by software developers. Begel et al. discuss some potential future directions for research in social media for software engineering in [8]. In [94] Storey et al. found how social media has revolutionized the way software development is done. In their recent work, Storey et al. have also investigated how usages of various social and communication channels affect software development [96]. Inspired by the guidelines and findings based on these works there have been many studies which have analyzed various social media channels. We describe some of the works below.

Stack Overflow, a social question and answering website related to software engineering questions, is one of the most researched social channels in software engineering research. Treude et al. examined the question asking and answering behavior of developers and their evolution in [108, 109]. User behavior in Stack Overflow has also been examined in several other studies such as [31, 7, 35, 10]. In [121] the authors studied the representation and social impact of gender in Stack Overflow. Barua et al. applied LDA to discover topics and trends present in questions and answers on StackOverflow [6]. The content in Stack Overflow has been analyzed in various other works such as [139, 118, 126, 22, 9, 124].

Another social media channel that has received a lot of attention is *GitHub*, a social coding website. Thung et al. [101] analyzed GitHub developer network to understand characteristics of influential projects and developers. In [42], the authors discussed the benefits and challenges of mining GitHub. How developers code socially and collaborate in GitHub has been evaluated in [111, 23, 112]. Vasilescu et al, collected thousands of projects from GitHub, in order to understand the relationship between context-switching and productivity of developers [120]. There have been other works which have evaluated the productivity of developers based on GitHub data such as [123, 66, 40, 78, 79]. Developer behavior across the platforms Stack Overflow and GitHub has been studied in [122, 5].

Developers' usage of microblogs such as *Twitter* has been explored by a number of prior studies. Singer et al. surveyed 271 and interviewed 27 active developers on Github [89]. They found that many developers are using Twitter to "keep up with the fast-paced development landscape". Specifically, developers use Twitter to get awareness of people and trends, extend their technical knowledge, and build connections with other developers. They also found that information overload, i.e., few useful information hidden in thousands of useless tweets, is one of the biggest challenges faced by developers in using Twitter.

A number of researchers have analyzed microblogs posted in Twitter (aka tweets) or built tools to support developers to better use Twitter for their day-to-day work. Bougie et al. analyzed 11,679 tweets posted by 68 developers from three open source projects [11]. They observed that software engineers leverage Twitter to communicate and share information. They also conducted a qualitative study on 600 tweets and group them into four categories: software related, gadgets and technological topics, non-technical topics, and daily chatter. Wang et al. analyzed 568 tweets posted by developers from the Drupal open source project [129]. They found that Drupal developers use Twitter to coordinate efforts, share knowledge, encourage potential contributors to join, etc. Tian et al. analyzed behaviours of software microbloggers on a large dataset that contains more than 13 million tweets posted by 42 thousand microbloggers [104]. They used 100 software related words to identify software related tweets and found that software related tweets often contain more URLs and hashtags, but fewer mentions than non-software related tweets. They also find that some of the microbloggers are very active on Twitter and have contributed many software related tweets. In another work, Tian et al. manually categorized 300 tweets that contain software related hashtags, e.g., "java", "csharp" into ten groups [103]. These ten groups include commercial, news, tools and code, question and answer, events, personal, opinion, tips, job, and miscellaneous.

The topics discussed on Twitter by end users of software, and the implications it can have for *requirements engineering* and *software evolution* has been explored in some recent works. Zampetti et al. did a quantitative analysis of about 153,853 URLs extracted from GitHub pull requests in order to analyze what external references are used by developers in documenting pull requests [140]. They found that the references to microblogs are quite low (0.68%). However, when they manually coded a small subset of pull requests, they found that URLs that refer to microblogging (e.g., Twitter) have a substantial impact on pull requests not directly related to source code, such as documentation updates or license updates. Mezouar et al.

proposed an approach to map tweets to bug fixing reports, and also did an empirical study the usefulness of Twitter in the bug fixing process [25]. They found that at least 33% of Firefox bugs and Chrome bugs can be discovered earlier by tracking the tweets from end-users, which may allow developers to discover and fix issues early. Guzman et al. have done an exploratory study about the potential of automating tweet analysis to support requirements engineering [32, 33]. Williams et al. have also explored the potential of Tweets for mining user requirements [133].

There have also been recent studies that explore some new social media channels. Macleod et al. did an analysis of how developers use screencasts to share and document software knowledge [55, 54]. An analysis of user comments on coding video tutorials present on YouTube was performed in [71]. Lin et al. recently studied how developers use *slack* to help them in their software development activities [50]. The developers' user of news aggregators such as *Reddit* and *HackerNews* has been explored in [3].

2.3 Mining Social Software Repositories

As discussed in Section 2.2 many studies have explored how developers use social media channels. Based on the outcome of these studies many techniques have been proposed which can make it easy for developers to use these media, and/or provide them with important insights which can help them in their software development activities. We discuss some of such works below.

Tag prediction for questions in Stack Overflow is a popular area in software engineering research and many approaches have been proposed for solving this problem [127, 144, 128]. Recently Uddin et al. proposed methods to identify opinionated sentences from Stack Overflow data and then mine aspects from such sentences [117, 116, 115]. Stack Overflow data has been used to build various automated tools to support software development [110, 14, 107, 63]. Techniques to mine knowledge graphs, and recommend analogical libraries based on Stack Overflow discussion were proposed in [16, 17, 18].

Many tools and techniques have been proposed to help developers better use GitHub. Thung et al. proposed a library recommendation system based on 1,008 GitHub projects [102]. Project and repository recommendation techniques were proposed in [142, 137, 138]. Techniques have also been proposed recently which can categorize projects and artifacts on GitHub [84, 53].

Researchers have also built tools to support developers to better use Twitter for their day-to-day work [1, 76, 85]. Achananuparp et al. build 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 [85]. Guzman et al. proposed ALERTme, an approach to automatically classify, group and rank tweets about software application [34]. Recently Sharma et al. proposed an approach to extract URLs from Twitter for software developers which can help in their learning and knowledge updates [87]. Recently some tools have been proposed to aid discovery and usage of *videos* related to software engineering. Ponzanelli et al. proposed an approach to extract relevant fragments from software development videos tutorials [73, 74, 72]. Text retrieval based tagging of software engineering video tutorials has been proposed in [26, 69].

.

2.4 Summary

In this chapter, we did a brief discussion of the works performed as a part of this dissertation. As seen in the previous sections social media channels have become imbibed in the day-to-day workflow of software development, and these channels support software developers in a number of ways. Twitter is also one of such social media and is quite popular among software developers. The works in this dissertation try to solve challenges that developers face while using Twitter for activities related to software development. Although the techniques developed in this dissertation primarily focus on Twitter, we believe that some of them should be generalizable enough for other social media channels also.

Chapter 3

Understanding Popular Software Engineering Content Produced in Social Networks

3.1 Introduction

Twitter is currently the most popular microblogging service in the world. Apart from using Twitter to connect with friends and family, people also use Twitter daily to share news and knowledge and discover latest information and updates about various topics of interest. Recent studies have found that software developers also use Twitter for their personal, as well as professional pursuits. Singer et al. [89] survey 271 GitHub developers and interview 27 of them to better understand their Twitter usage. They find that software developers use Twitter quite extensively in their professional activities. Developers use Twitter to stay aware of the latest software trends and practices, to extend their software knowledge by learning new stuffs and to maintain relationships with fellow software developers.

In this work, we extend Singer et al.'s study by investigating *events* in software engineering Twitter space. An event corresponds to a set of topically-coherent microblogs that are shared by many Twitter users at a point in time. It can be viewed as a popular *trending topic* in the software engineering Twitter space happening at a particular point in time. By studying events, we can discover *hot* topics that interest many developers. Different from Singer et al.'s work which focuses on getting insight of developer's use of Twitter by interviewing developers, this work analyzes contents of tweets about popular topics that developers generate. Moreover, while Singer et al. reported that developers use Twitter to get up-to-date with the latest trends and consume knowledge, this work drills deeper by investigating the kinds of popular trends and knowledge that get disseminated widely.

To find events, we first monitor a set of 90,883 users who are potentially interested in software development. We collect microblogs that are generated by these users over an 8 month period which amounts to 48,889,030 microblogs in total. Next, we need to identify software engineering related tweets among these 48.9 million tweets. Unfortunately, this will require a prohibitive amount manual effort since automated solutions still cannot identify software engineering tweets with high accuracy (c.f., [85, 76]). To make the identification of software engineering related tweets practical, in this work we only study microblogs that mention each of the following popular programming languages, i.e., C#, Java, Python, Scala and Ruby. We leave the study of other software engineering related tweets as future work. We then apply a state-of-the-art Twitter event detection algorithm [24] on each of the five sets of microblogs to find events related to each of these programming languages.

We sort the identified events based on their popularity (i.e., number of tweets involved in the event), and selected the top 200 events for each of the programming language. These 1,000 events are then manually analyzed using the open coding procedure [97, 82] to create event categories. We then investigate three research questions: 1) What are some hot software engineering related events in Twitter space? 2) What are the categories of software engineering related events in Twitter space? 3) How hot is each event category?

Our study is the first step towards a deeper understanding of tweets that interest developers. A good understanding of interesting tweets will guide our future work on the construction of a recommendation system that can highlight fresh, relevant, and interesting pieces of information from the Twitter stream to keep developers up-to-date with recent trends and gain new knowledge. Such a solution will address challenges that prevent developers from using Twitter, e.g., information overload, etc. [89].

The contributions of this work are as follows:

1. We are the first to investigate events or trending topics that appear in the software engineering Twitter space.

2. We perform an open coding procedure on 1,000 events to group them into categories, and answer three research questions that shed light to topics that interest many people in software engineering Twitter space.

The structure of the remainder of the chapter is as follows. In Section 3.2 we give some background on Twitter and the event detection approach used in this work. In Section 3.3, we describe the methodology that we follow in this exploratory study. We describe findings of our study in Section 3.4. Finally, we conclude and mention future work in Section 3.5.

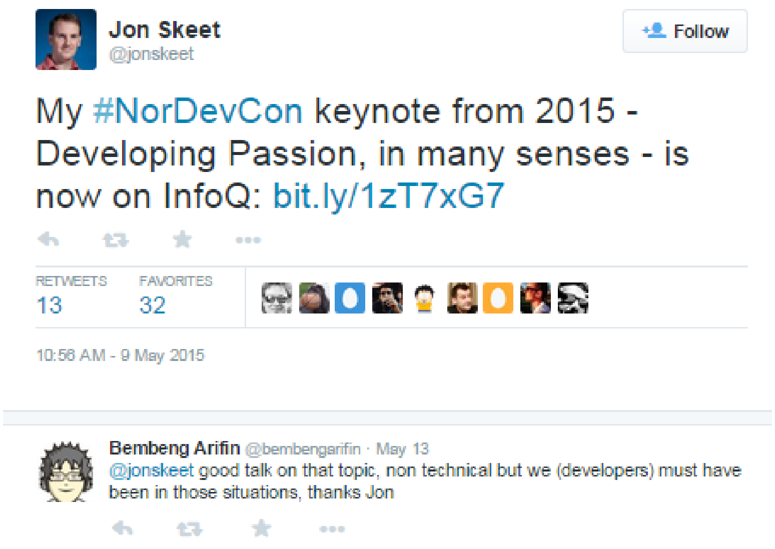
3.2 Background

In this section, we briefly describe Twitter and the state-of-the-art NLP algorithm that we use for detecting events in Twitter.

3.2.1 Twitter

Twitter is the most popular microblogging and social media platform with more than 300 million active users generating more than 500 million microblogs daily. Each registered user in Twitter can post microblogs (popularly known as tweets) with a maximum length of 140 characters. As the amount of text that users can post is limited, users can include in their tweets URL links to the actual blogs, articles, news, etc., which contain detailed information about the topic that the users are intended to share. Twitter supports the concept of hashtags; a user can use the symbol "#" in front of a word to emphasize that the tweet being posted is associated with a particular topic described by the word.

Twitter also allows one user to follow other user with the former (known as a follower) subscribing to all the tweets of the latter (followee). Instead of sharing their original tweets, a user can also share tweets made by others by *retweeting* existing tweets. This retweeting process broadcasts an existing tweet received by a user to all his/her followers. Additionally, users can *reply* to other users' tweets or *favourited* other users' tweets. Furthermore, while writing a tweet a user can explicitly *mention* another user by using the "@" sign followed by the user's username. Tweets mentioning a user will be forwarded to the user.


Jon Skeet
@jonskeet
Follow

My #NorDevCon keynote from 2015 - Developing Passion, in many senses - is now on InfoQ: bit.ly/1zT7xG7

RETWEETS 13
FAVORITES 32

10:56 AM - 9 May 2015

Bembeng Arifin @bembengarifin · May 13
@jonskeet good talk on that topic, non technical but we (developers) must have been in those situations, thanks Jon

Figure 3.1: A Sample Microblog (i.e., Tweet) on Twitter.

Figure 3.1 shows a tweet which was posted by user "jonskeet". The sample tweet contains a hashtag *#NorDevCon*. This tweet also shares an URL, i.e., `http://www.infoq.com/presentations/developer-passion`, which points to a webpage where detailed information about the user's keynote is made available. The tweet has been *retweeted* 13 times and *favourited* 32 times.

Also, some users (e.g., “bembengarifin”) have *replied* to the tweet.

3.2.2 Event Detection in Twitter

Event detection in Twitter is becoming a hot research topic recently, as it can help individuals or organizations to better understand what is trending from large real-time data. In this work, to detect software-related event on Twitter, we select one of the latest and promising Twitter event detection approach proposed by Diao and Jiang [24], and apply it on software-related tweets.

Diao and Jiang design a unified model which combines a topic model to model user interest, a dynamic non-parametric model to model events, and a probabilistic matrix factorization component to capture the relationship between events and topics. The unified model tries to separate personal from event-related tweets, identify tweets that belong to the same event, and penalize long-term events since most events are short-lived. It accepts as input a stream of tweets and produce a series of events along with tweets that belong to each event.

3.3 Proposed Approach

Figure 3.2 shows the methodology we follow in our study which contains 3 major steps: Twitter Data Extraction, Event Identification, and Open Coding.

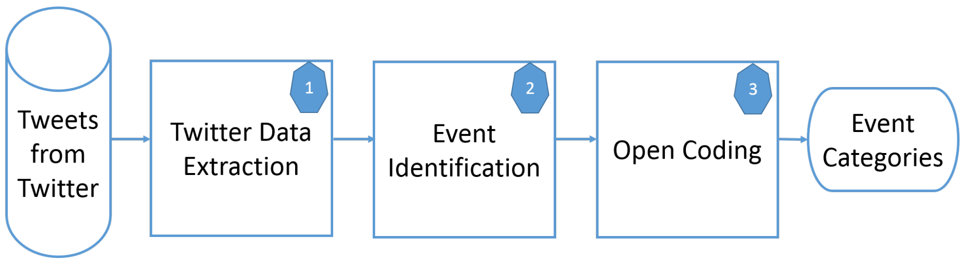


Figure 3.2: Empirical Study Methodology

In Step 1, we identify a set of 90,883 Twitter users who are potentially interested in software development. This identification was done following the methodology used in our previous work [104, 85]. We start with a seed set of 100 users who are well-known in software development[1] and include other users that follow or are followed by at least five of the seed users. We then periodically crawl all tweets of these users for an 8 month period between September 2012 to April 2013 and we collect in total of 48,889,030 tweets. From these tweets, we identify 5 sets of tweets; each set consists tweets that mention one of the following 5 programming languages: C#, Java, Python, Ruby and Scala. After this keyword filtering process, we have sets of C#, Java, Python, Ruby, and Scala tweets of sizes 27,102, 117,385, 54,862, 104,528 and 35,634 respectively. At the end of this step, we order tweets in each set based on the time they were posted.

In Step 2, for each series of tweets for a programming language extracted in Step 1, we identify events in them by running a state-of-the-art Twitter event detection technique by Diao and Jiang [24]. Diao and Jiang design a unified model which combines a topic model to represent user interest, a dynamic non-parametric model to represent events, and a probabilistic matrix factorization component to capture the relationship between events and topics. The unified model tries to separate personal from event-related tweets, identify tweets that belong to the same event, and penalize long-term events since most events are short-lived. The precision of Diao and Jiang's technique has been shown to be high (precision@5=100% and precision@30=90%). After processing each of the 5 series of tweets, the technique produces a ranked list of the events and we take the top-200 events sorted based on their sizes (i.e., the number of tweets in the event). At the end of this step, we have a set of 1,000 events that we need to group into categories.

In Step 3, we follow the open coding procedure [97, 82] to generate event categories. Open coding is performed in three iterations. In the first iteration, each event and tweets contained in it is read and a short code (i.e., description) is assigned to it. In the second iteration, the codes are analyzed to create higher-level concepts by merging similar codes together. In the final iteration, the concepts are analyzed

[1]`http://noop.nl/2009/02/twitter-top-100-for-software-developers.html`

to create a small set of categories. After the categories have been identified, the categories are sorted based on how "hot" they are (i.e., how many events belong to each category).

3.4 Experiments & Analysis

Our study aims at answering the following three research questions. This knowledge can be used to build an automatic recommendation system which can find interesting software engineering related events from Twitter stream.

3.4.1 RQ1: What are some hot software engineering related events in Twitter space?

By answering this question, we want to highlight some of the hot or popular software engineering events for the time period we consider. Table 3.1 shows sample hot events found for each of the 5 languages we consider.

The example hot event for Java is a security bug that affected Java based web browsers in Jan 2013 which was shared by at least[2] 369 Twitter users. This was an important advisory to developers as well as general public to disable or not use Java based web browsers until the issue is resolved. For C#, the hot event is the release of a viral blog specifying benefits of using C# for mobile development, which was shared by at least 181 Twitter users. This blog may inspire many developers who work on mobile development to use C# rather than other languages. For Python, it is a trademark dispute which was shared by at least 382 Twitter users. This dispute was a call-to-arms for Python developers and enthusiasts to join forces in a legal battle to keep the name Python. For Ruby, it is the release of Ruby 2.0.0, an event that many Ruby developers were likely to be waiting for, and this event was shared by at least 842 Twitter users. For Scala, the joining of Rod Johnson (creator of

[2]We only monitor a subset of all Twitter users.

Table 3.1: Hot Software Events for Each Programming Language

Language	Date of First Tweet	Event Description	Sample Tweets	Tweet Count
Java	11/01/2013	Security vulnerability found in Java	•Feds warn PC users to disable Java •security vendors warn users to disable java after zero day exploit is found	369
C#	02/01/2013	A blog specifying reasons why C# is the best language for mobile development was posted.	•post by xamarin why c# is the best language for mobile development •They've got a horse in the race, but yes. RT @xamarinhq: Eight reasons C# is the best language for mobile development	181
Python	14/02/2013	A company in United Kingdom applied to trademark "Python" for all software and services.	•unbelievable, some random software company in Europe is trying to trademark "Python" •Python trademark at risk in Europe: We need your help!	382
Ruby	24/02/2013	Ruby 2.00 was released	•Ruby 2.0.0-p0 was released •Ruby 2.0.0-p0 is released Come and get it! Boosts to language support, performance, debugging, and built in libs.	842
Scala	01/10/2012	Rod Johnson, creator of Spring Framework in Java joining Typesafe Inc.(founded by authors of Scala team)	•Excited to be getting involved with @typesafe. I love Scala more and more and it's a gr8 team with Martin, Jonas & crew & now Mark Brewer •Proud to welcome Rod Johnson (@springrod) to the Typesafe board: @typesafe #akka #scala #playframework	150

Java Spring framework) to Typesafe Inc. (the firm mainly responsible for pushing Scala's commercial adaptation) is a hot event. This event was shared by at least 150 Twitter users. This event was an exciting news for Scala developers and it may motivate many other developers to learn and use Scala.

3.4.2 RQ2: What are the categories of software engineering related events in Twitter space?

In this research question we want to group software engineering events into categories. Following the open coding procedure described in Section 3.3, we have been able to determine 11 categories of events (10 main categories + "Others") in Table 3.2.

Tweets occurring in category *Article and Multimedia Sharing* such as: "*functional programming principles in scala starts again next monday still time to enroll*" helps interested developers by exposing them to learning resources available. For category *Technical Discussion* tweets such as: "*Scala protip lazy val is not* × *free (or even cheap). Use it only if you absolutely need laziness for correctness, not for optimization*" can help developers in their programming activities. Tweets like: "*Feels so good @ScalaIDE: Scala IDE 3.0.0 is out With semantic highlighting, Scala debugger*" occurring in *New Releases* category can be extremely helpful for developers who are on lookout for such tool. "*Dropbox Hires Away Googles Guido Van Rossum, The Father Of Python*" is an example of a tweet in the category *News*, which the users may find interesting. Tweets such as: "*ebook dealday think python $1599 save 50% use code deal*" in the category *Product Promotions* help developers to be aware of latest deals and promotion on books and products they might be interested in. "*I'll be kicking off Build with style and rocking C# on 2.5 billion devices at @xamarinhqs #bldwin Welcome Party!*" is an example of tweet in the category *Community events*. This was used to attract and encourage software developers to join *Microsft Build Devekoper Conference*. Tweets in *se-*

curity updates category such as "*sql injection vulnerability in ruby on rails affects all versions*" help to quickly disseminate security related information to developers. Additionally, tweets such as: "*Seeking Contributors for the Facebook C# SDK*" in the *Crowdsourcing Request* category can be extremely helpful for developers looking for such opportunities. Similarly tweets that fall under the *Career* category can be helpful to software developers hunting for jobs, while tweets that fall under the *Satires* category can help developers to de-stress.

3.4.3 RQ3: How hot is each event category?

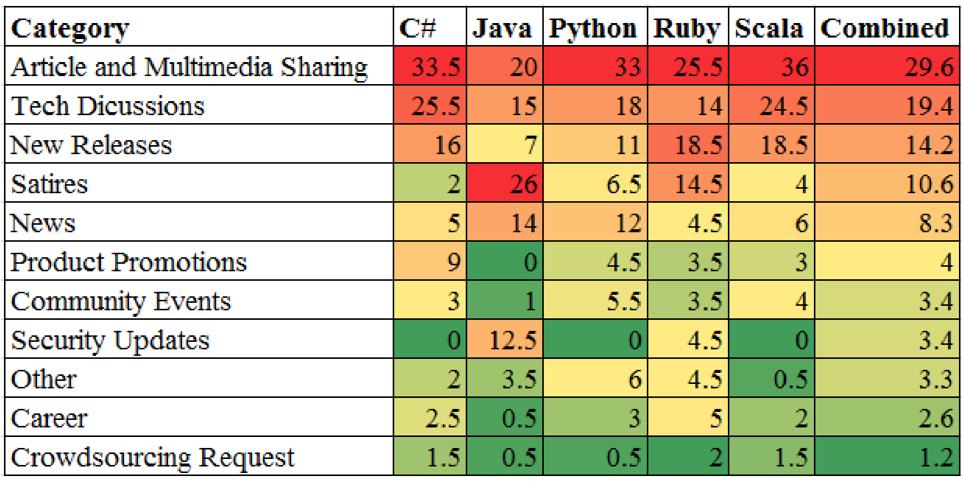
In this research question, we analyze the popularity of each event category. For each event category, we count the total number of events in the category. We then plot a heat map showing the hottest event categories (categories with the most tweets) for each programming language and overall in Figure 3.3.

From the figure, we can note that the top-3 hottest event categories are: *Article and Multimedia Sharing, Technical Discussions*, and *New Releases*. To help developers keep up-to-date with recent trends, we encourage future studies to build automated solutions which are able to find, recommend, and summarize tweets that fall under the ten categories, especially the hotter ones.

Another interesting observation to note is that for the 8 month period, popular *security updates* occurred only for Java and Ruby. Java security updates are retweeted by a large number of Twitter users and one eighth of all popular Java event tweets are about security updates. Another observation is that the number of popular *community events* are higher for Python, Ruby and Scala as compared to C# and Java. Another thing to notice is that for Java, the *Satire* category is more popular than the other languages, on the other hand, no events occur in the *Product Promotions* category.

Table 3.2: Categories of Events in Software Engineering Twitter Space

Category Name	**Description**
Article and Multimedia Sharing	Tweets sharing articles, blogs, tutorials, or videos related to software development.
Technical Discussions	Tweets discussing some technical issues related to software development.
New Releases	Tweets announcing the release of a new software version, tool, etc.
Satires	Tweets sharing jokes and funny quotes generally related to software bugs or issues.
News	Tweets sharing news items related to software development such as joining of a new CEO for a large software company, etc.
Product Promotions	Tweets promoting commercial books and tools related to software development.
Community Events	Tweets about conferences, coding events, anniversaries, etc.
Security Updates	Tweets about latest security issues and fixes affecting software products and frameworks.
Career	Tweets about job openings and candidates sharing their availability for hire.
Crowdsourcing Requests	Tweets requesting users to contribute to open source projects, surveys, petitions, etc. related to software development.
Others	All other events which do not fall into one of the above categories



Category	C#	Java	Python	Ruby	Scala	Combined
Article and Multimedia Sharing	33.5	20	33	25.5	36	29.6
Tech Dicussions	25.5	15	18	14	24.5	19.4
New Releases	16	7	11	18.5	18.5	14.2
Satires	2	26	6.5	14.5	4	10.6
News	5	14	12	4.5	6	8.3
Product Promotions	9	0	4.5	3.5	3	4
Community Events	3	1	5.5	3.5	4	3.4
Security Updates	0	12.5	0	4.5	0	3.4
Other	2	3.5	6	4.5	0.5	3.3
Career	2.5	0.5	3	5	2	2.6
Crowdsourcing Request	1.5	0.5	0.5	2	1.5	1.2

Figure 3.3: Hotness of event categories across languages (red = most popular, green = least popular). Numbers represent the ratios of the total number of tweets of an event category to the total number of tweets (in percentages).

3.4.4 Threats to Validity

Similar to other exploratory studies, there are some threats that may affect the validity of our study. First, we only study 90,883 Twitter users and their 48,889,030 microblogs which we collect over an 8 month period. Second, the event detection algorithm by Diao and Jiang [24] may wrongly identify events. Third, we only manually analyze 1,000 events using the open coding procedure. Fourth, the open coding procedure involves subjectivity and most of the labeling decisions are made by one person. Still, 48 million (microblogs) and 1,000 (events) are large numbers. Also, Diao and Jiang's algorithm is a state-of-the-art algorithm and has been shown to perform well. Furthermore two researchers other than author reviewed the author's labels to improve them.

3.5 Conclusion

Today, Twitter is one of the most popular mediums for information and resource sharing. Software developer also use Twitter a lot in their career related activities especially for remaining updated with the latest happenings, gaining new knowledge, and maintaining a community network [89]. In this work, we perform an exploratory study of events (or trending topics) in software engineering Twitter space which have attracted the interest of many developers. We collect more than 48 million tweets made by close to 90,000 Twitter users over a period of 8 months, filter them based on 5 programming language names, and identify events by running a state-of-the-art Twitter event detection algorithm [24]. We manually analyze 1,000 identified events and group them into categories using the open coding procedure. At the end, we analyze how hot each of these event categories is. Our exploratory study shows that most events that attract the interest of many Twitter users relate to: article and multimedia sharing, technical discussion, new releases, satires, news, product promotions, community events, security updates, career, and crowdsourcing request. The first three categories in particular are the hottest ones.

In the future, we plan to expand this study to include tweets about other programming languages, libraries, software development methodologies, etc. in order to gain a good understanding of features of noteworthy tweets in the software engineering Twitter space. Our eventual goal is to build a recommendation system that can identify tweets that interest many developers (e.g., tweets that fall into categories identified in this study) to help developers keep up-to-date with recent trends and learn new knowledge from Twitter stream. Such a solution will help solve challenges that prevent developers from using Twitter, e.g., information overload, etc. [89], potentially resulting in an increased adoption of Twitter to improve software development activities.

Chapter 4

Automatic Identification of Software Relevant Content in Social Media

4.1 Introduction

Twitter, as one of the largest and popular on-line social network sites, provides a platform to let people share news, disseminate opinions, and connect with one another. It is growing fast in the recent years and is reported to have more than 600 million registered users. Along with the rapid increase in the number of users, there is also a dramatic increase in the number of microblogs (i.e., tweets) posted every day. Many Twitter users who *follow* a large number of other users[1] receive thousands of tweets daily. This causes an information overload problem which makes it hard for users to find relevant and interesting tweets among the mass of tweets that they receive.

A large number of software developers also use Twitter quite frequently, even for their professional activities, e.g., to share and obtain latest technical news, to support project and community management, etc. Unfortunately, as is the case with other normal Twitter users, they find hard to extract useful information from tweets, especially those that can help them in their professional activities. Singer et al. sur-

[1]In Twitter, a user will receive tweets generated by users that they *follow*.

veyed 271 and interviewed 27 active developers and found that although developers are using Twitter for their professional activities and development, they often find it a challenge to deal with the many irrelevant tweets (i.e., noise) in their Twitter streams [89]. Bougie et al. found that many of the tweets that are generated even by software developers are related to "daily chatter" [11]. Indeed, it is common for people to spread personal yet inconsequential information in Twitter, e.g., "Yay! Today is Friday", "It's cloudy today", etc. To make Twitter a better tool for software engineers, there is a need for a technique that can help developers identify software related tweets from the mass of other tweets. This automated approach should be able to prioritize or rank tweets for software developers' professional use (e.g., getting latest technical news) so that more relevant tweets (i.e., software related tweets) can be ranked higher than irrelevant tweets (e.g., daily chatter). Additional benefits of identifying software related tweets are elaborated in Section 4.2.1.

Prior studies on Twitter have proposed two basic approaches to identify software related tweets. One is a support vector machine (SVM) based approach proposed by Prasetyo et al. that requires a training set of tweets labeled as software related and non software related [76]. Unfortunately, building a representative set of tweets for training an SVM classifier requires much manual effort and to the best of our knowledge no such representative training data is available till date. Another approach is a keyword based approach proposed by Achananuparp et al. and Tian et al. that takes as input a list of software related keywords and identifies a tweet as software related if it contains at least one of the keywords [1, 104]. Both Achananuparp et al. [1] and Tian et al. [104] make use of a list containing 100 software related words. However, this list is not comprehensive and many software related tweets do not contain any of the 100 words. Furthermore, software related contents on Twitter might change over time. Unfortunately, it is hard for both the SVM and keyword based approaches, which rely on a static set of training data or a list of keywords, to keep with this change without much effort (e.g., continuous labeling effort). Additional limitations of the existing approaches are elaborated in Section 4.2.2.

To deal with the limitations of existing approaches, in this work we propose NIRMAL, which is a language model based approach to identify software related tweets. A language model can be regarded as a statistical tool to capture regulations in a text corpus. It is widely used in natural language processing (NLP) area to help machine translation, speech recognition, etc. [141, 59, 47, 57]. Recently, it was also adopted by researchers in software engineering to capture the naturalness of source code and support code suggestion and completion tasks [36, 65, 2, 113]. With a language model, we can take a corpus (i.e., a set of documents), model its regularities and use the resultant model to predict if another document is related to the documents in the corpus. To build a language model, our approach namely NIRMAL takes a large number of contents from StackOverflow, the largest site for developers to post questions and get answers. By doing this, we can capture the regularities among documents that are software related. Given a new tweet, we calculate the probability of the tweet to be software related, using the model learned from StackOverflow content. The probability calculated corresponds to the computation of the similarity between the given tweet and the software related contents on StackOverflow. If a tweet receives a higher probability using the model, it means that it has a higher chance of being a software related tweet. To improve the performance further, we have also extended a standard language model by considering the repetitiveness of contents in a tweet that can often differentiate between informative tweets and meaningless tweets.

Note that compared to the previous two approaches (i.e., SVM based [76] and keyword based [1, 104]), our approach is more useful as it does not require a representative set of manually labeled tweets which takes much manual effort to create, and it does not suffer the same limitations as the keyword based approach. To capture new trends and developments in software development, the language model can also be incrementally updated with new contents from StackOverflow easily ,which requires no/little manual effort (i.e., no manual labeling effort is needed).

To evaluate the effectiveness of our approach, we have trained NIRMAL on a large set of contents from the StackOverflow data dump. We then used NIRMAL to rank 6,294,015 tweets posted by 90,883 micro bloggers that we had collected in April 2013. We manually evaluated the top-ranked tweets and determined whether they are software related or not. The results were evaluated using a measure accuracy@K, which is defined as the proportion of tweets in the top-K positions that are software related. The metric accuracy@K has also been used to evaluate past studies such as [38, 130, 136]. We found that NIRMAL can achieve an accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 of 0.900, 0.820, 0.720, 0.707 and 0.695, respectively. On the other hand, a random model can only achieve an accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 of 0.400, 0.280, 0.280, 0.220 and 0.240, respectively. Thus, NIRMAL can improve the random model by 125%, 192.86%, 157.14%, 221.21% and 189.58%, in terms of accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 respectively. We have also integrated our approach with the keyword based approach by Achananuparp et al. and Tian et al. and found that we can improve the accuracy@10, accuracy@50, accuracy@100, accuracy@150, and accuracy@200 of the keyword based approach by 11.11%, 31.43%, 28.38%, 28.32% and 29.14%, respectively.

The contributions of this work are as follows:

1. We propose a new approach, named NIRMAL, that can automatically identify software related tweets. Our approach makes use of a language model to capture the regularities of software related documents by leveraging the mass of data available in StackOverflow. Our approach also measures the repetitiveness of contents to differentiate between meaningful tweets and meaningless ones. Different from the existing approaches, NIRMAL does not require a representative training set of labeled tweets or a long list of representative keywords.

2. We have used NIRMAL to rank 6,290,415 tweets from 90,883 microbloggers

that were collected in April 2013. The experiment results show that NIRMAL can achieve a high accuracy@K scores (i.e., up to 0.900) and also improve the keyword based approach by up to 31%. We have also investigated the impact of different settings to determine the effectiveness of NIRMAL.

The structure of the remainder of this chapter is as follows. In Section 4.2, we elaborate the motivation of our work further. In Section 4.3, we present the background information about Twitter and language modeling. In Section 4.4, we describe our proposed language-based approach that can automatically identify software related tweets. In Section 4.5, we present our experiment settings and the results of our experiment. We finally conclude and mention future work in Section 4.6.

4.2 Motivation

In this section, we first describe the benefits of identifying software related microblogs in more detail. We then elaborate limitations of the two basic approaches that have been used to extract software related tweets and how these limitations are addressed by NIRMAL.

4.2.1 Why identify software related tweets?

As microblogging services have become very popular in recent years, more and more developers are using microblogs to share news and connect with one another. Software engineering researchers also noticed this trend among developers, and have started to analyze how do microbloggging sites, e.g., Twitter, help developers in their professional activities. Several studies have analyzed the contents of microblogs that developers post on Twitter [11], investigated behaviors of software microbloggers on Twitter [129, 104], and surveyed developers on how they use Twitter [89].

Researchers who conducted the above studies found that developers indeed use Twitter a lot to support their professional activities by sharing and discovering various information from microblogs, e.g., new features of a library, new methodologies to develop a software system, opinions about a new technology or tools, etc. They also find that developers use Twitter to post contents to support project management and coordinate activities inside a community. However, software microbloggers also post a lot of microblogs that are not relevant to software development, e.g., microblogs about non-technical news, personal events, jokes, etc. Since software microbloggers are creating some amount of software related knowledge together with a larger amount of non software related contents, it becomes a challenge to discover interesting software related information from microblogs that a developer receives. In fact, this is reported as one of the major challenges faced by software microbloggers who are using Twitter and is one of the barriers to the adoption of Twitter [89]. Therefore, an automated approach that can identify interesting microblogs, e.g., software related tweets, is needed.

Besides the above-mentioned practical need, automatic identification of software related tweets, can also open up a new avenue of research: it can be used to extract a new type of software repository that can be mined to support various software development and evolution tasks. Some potential tools that can be built from the identified software related tweets include:

1. Tools that discover and visualize trends of software related contents on Twitter.

2. Tools that recommend contents on Twitter that are specific to a developer's specific needs and interests.

3. Tools that mine opinions about APIs, IDEs, programming languages, technical solutions, etc., from contents on Twitter.

4.2.2 Why a new language model based approach?

In the literature, researchers have proposed two basic approaches to identify software related microblogs, i.e., Support Vector Machine (SVM) based approach and keyword based approach. We elaborate the details of these two existing approaches as well as their limitations below:

SVM based approach. Prasetyo et al. proposed to use SVM to predict if a tweet is software related or not [76]. They manually labeled 300 tweets as either software related or not and used a part of the labeled tweets as a training data to learn a classifier using SVM, and applied the classifier on another set of tweets to predict whether they are software related or not. However, their approach only considered 300 tweets from the millions of tweets. To generalize the SVM based approach, researchers need to label a large and representative sample of tweets on Twitter, which takes a lot of time. The 300 tweets are selected by checking the presence of nine software related hashtags, and therefore they have a nearly balanced data set: 47% of the tweets are software related while the other 53% are non software related. In reality, the majority of tweets do not have hashtags and there are much more non software related tweets than software related tweets. The extremely unbalanced data is likely to impact the effectiveness of the SVM classifier. In addition, contents on Twitter are evolving as new technologies and tools are introduced to the market; this means that the model might need to be updated based on new labeled tweets. Unfortunately, this would require a continuous effort to label new tweets as either software related or not which would be costly.

Keyword based approach. Achananuparp et al. and Tian et al. used a list of keywords to identify software related tweets [1, 104]. Different from the SVM based approach, the keyword based approach does not require labeled tweets. It simply takes a set of software related words as input and identifies a given tweet as software related if it contains any of the words in the provided set of words. Unfortunately, there are a number of limitations with this approach. First, it is hard to construct a comprehensive list of software related words that could identify whether a tweet is software related or not. For instance, tweet *How To: Use the Entity Framework Designer* `http://t.co/SteQkWAKfN` is talking about a new resource that a developer wants to share with other developers, however it does not contain any of the keywords considered by Achananuparp et al. and Tian et al. Second, some words can have multiple meanings and not all of the meanings will be software related, e.g., Java, eclipse, etc. The problem is aggravated with the fact that tweets can be written in multiple languages. In English, a word might correspond solely to a software related concept; however, it can correspond to a completely unrelated concept in another language (e.g., Ada is a programming language and the same word means "there is (are)" in Indonesian). Third, similar to the SVM based approach, the keyword based approach cannot automatically update itself when new technologies or tools are introduced. Someone needs to manually update the list of keywords to make the approach adapts to new technological updates.

To address the challenges faced by the existing two approaches, we propose a new approach namely NIRMAL to identify software related tweets leveraging language model learned from StackOverflow. The benefits of NIRMAL include: 1) it does not require labeled software related and non software related tweets, 2) it does not require manually defined keyword list, 3) it takes the context of a word into consideration. We describe our approach in detail in Section 4.4.

4.3 Background

In this section, we first describe the two platforms that we consider in this study, namely Twitter and Stack Overflow. We then provide a brief introduction of language model.

4.3.1 Twitter

Twitter is the most popular and largest microblogging site worldwide. It already has more than 600 million registered users generating over 500 million tweets daily [2]. The number of active Twitter users is growing rapidly, it increased from around 167 millions in the 3rd quarter of 2012 to 284 millions in the 3rd quarter of 2014. [3]

Twitter allows a user to post text messages, referred to as "tweets", with a maximum length of 140 characters. To address the limitation on the length of tweets, many Twitter users include url links in their tweets pointing to webpages such as blogs, news, etc. that contains more information. Twitter users can also include hashtags, which typically start with a "#" symbol. If a user clicks on a hashtag, Twitter will show other tweets with the same hashtag. In Twitter, one can *follow* another user; a user (follower) who follows another user (followee) will subscribe to all tweets that are posted by the followee. Besides composing new tweets, Twitter allows users to perform other activities. This includes *retweeting* an existing tweet that are posted by other users. A user that retweets a tweet will broadcast the tweet to all of his/her followers. Retweets typically start with the keyword "RT". Users can also reply to an existing tweets or tweeting directly to a user. A reply or direct tweet contains "@Username" to identify the user the tweet is intended for. Twitter also supports users to favorite a tweet to show their interest in the content of a tweet.

Figure 4.1 shows a sample tweet posted by "C# Corner". The sample tweet contains two hashtags, i.e., csharp and csharpcorner. This tweet specifies another Twitter user using the "@" symbol. It also contains a url link that points to a blog that talks about how a number can be converted to a string in C#.

4.3.2 Stack Overflow

Stack Overflow, created in 2008, is the most popular question answering sites specially designed for developers. Stack Overflow provides a platform for developers

[2]https://about.twitter.com/company

[3]http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

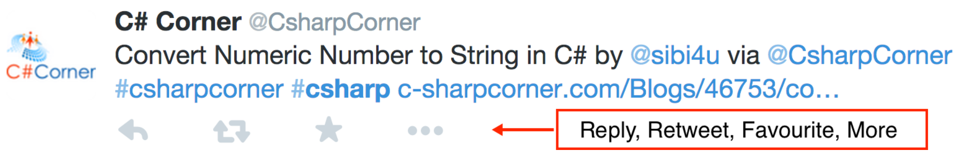

C# Corner @CsharpCorner
Convert Numeric Number to String in C# by @sibi4u via @CsharpCorner #csharpcorner #csharp c-sharpcorner.com/Blogs/46753/co...

Figure 4.1: A sample microblog (i.e., tweet) on Twitter.

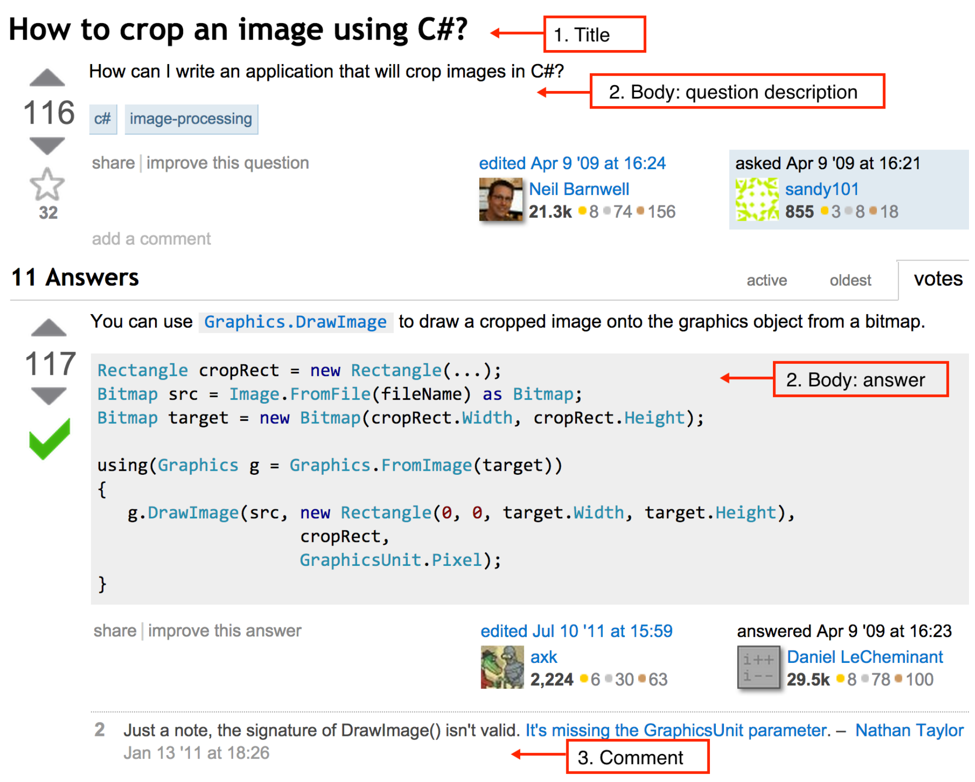
to help one another by asking and answering software related questions. It has become a large knowledge source with more than 2 million registered users contributing over 7 million questions.[4] The large amount of software related question-and-answer threads in Stack Overflow is a good source of information to mine and study. Past research works have used Stack Overflow data to: discover development topics and trends [6], build software-specific word similarity database [105], automatically generate code comments [134], etc.

Figure 4.2 shows a sample question-and-answer thread extracted from Stack Overflow. Each question-and-answer thread in Stack Overflow contains three types of information: title, body, and comments. The title of a thread is a short summary of the question. The body of a thread contains the description of the question and one or more answers if the question has been answered. The comments of a thread could be comments to the question or comments to any of the answers. These three different types of contents have different properties: title and comments contain more natural language text, while body is usually a mixture of text and pieces of code. Furthermore, comparing title and comments, title usually contains more technical words while comments might contains some non technical sentences or phrases, such as “thank you”, etc.

4.3.3 Language Model

A statistical language model is a probability distribution over word sequences. It assigns a probability to any sequence of words to present the likelihood of the se-

[4]http://en.wikipedia.org/wiki/Stack_Overflow#cite_note-soUSERS-17



How to crop an image using C#?

116

How can I write an application that will crop images in C#?

c# image-processing

32

share | improve this question

edited Apr 9 '09 at 16:24
Neil Barnwell
21.3k ●8 ●74 ●156

asked Apr 9 '09 at 16:21
sandy101
855 ●3 ●8 ●18

add a comment

11 Answers

active oldest votes

117

You can use `Graphics.DrawImage` to draw a cropped image onto the graphics object from a bitmap.

```
Rectangle cropRect = new Rectangle(...);
Bitmap src = Image.FromFile(fileName) as Bitmap;
Bitmap target = new Bitmap(cropRect.Width, cropRect.Height);

using(Graphics g = Graphics.FromImage(target))
{
   g.DrawImage(src, new Rectangle(0, 0, target.Width, target.Height),
                    cropRect,
                    GraphicsUnit.Pixel);
}
```

share | improve this answer

edited Jul 10 '11 at 15:59
axk
2,224 ●6 ●30 ●63

answered Apr 9 '09 at 16:23
Daniel LeCheminant
29.5k ●8 ●78 ●100

2 Just a note, the signature of DrawImage() isn't valid. It's missing the GraphicsUnit parameter. – Nathan Taylor
Jan 13 '11 at 18:26

Figure 4.2: A sample question-and-answer thread on Stack Overflow.

quence occurring in the language it models. For example, a good language model learned from English corpus will assign higher probability scores to sentences in English than sentences in other languages.

More formally, given a word sequence $S = t_1 t_2 t_3 \dots t_n$, a language model estimates the probability of this sequence to be represented by the model as:

$$P(S) = P(t_1) \prod_{i=2}^{n} P(t_i | t_1, \dots, t_{i-1}) \tag{4.1}$$

In the Equation 4.1, the probability for a sequence S is defined as a product of a series of conditional probabilities. Conditional probability $P(t_i | t_1, \dots, t_{i-1})$ represents the likelihood that word t_i follows the words that appear before it (i.e., $t_1, \dots, t_{i-1}$).

In practice, it is not practical to store all $P(t_i | t_1, \dots, t_{i-1})$ since there is a huge number of possible prefixes. Therefore, researchers have proposed methods to simplify this probability by including some assumptions. N-gram language model is one of such simplifications, which has been proven to be effective in practice. The N-gram language model assumes that the probability of a word t_i to appear after a series of words $t_1, \ldots, t_{i-1}$ could be estimated by considering only the previous $N-1$ words rather than all previous words. More formally, the following equality is assumed.

$$P(t_i | t_1, \ldots, t_{i-1}) = P(t_i | t_{i-N+1}, \ldots, t_{i-1}) \tag{4.2}$$

The probability on the right hand side of Equation 4.2 can be estimated from a training corpus (i.e., a set of textual documents) by computing the ratio of the number of times word t_i follows the prefix sequence $t_{i-N+1}, \ldots, t_{i-1}$ and the number of times the prefix sequence $t_{i-N+1}, \ldots, t_{i-1}$ appears in the training corpus. More formally, we can compute the probability as follows:

$$P(t_i | t_{i-N+1}, \ldots, t_{i-1}) = \frac{count(t_{i-N+1}, \ldots, t_{i-1}, t_i)}{count(t_{i-N+1}, \ldots, t_{i-1})} \tag{4.3}$$

Consider the sample tweet shown in Figure 4.1 which is a sequence of words. If we use a bigram language model (N=2), the probability of "convert numeric number to string in C#" could be calculated as

$$\begin{aligned}P(convert, numeric, number, to, string, in, C\#) = \\ P(convert|\langle s\rangle)P(numeric|convert)P(number|numeric) \\ P(to|number)P(string|to)P(in|string) \\ P(C\#|in)P(\langle/s\rangle|C\#)\end{aligned}$$

In the above equation, $\langle s\rangle$ denotes the start-of-sentence marker and $\langle/s\rangle$ denotes the end-of-sentence marker. Since the probability of each word in a sentence is often small, and the multiplication of many small numbers can cause underflow problem, rather than computing the probability of a sentence, the logarithm of this probability is often computed. The negation of this logarithm normalized by the number of words is often referred to as the *perplexity* of the sentence. In this work, we denote the perplexity of a sentence S as $PP(S)$. It is defined more formally below:

$$PP(S) = \frac{-1}{n} log(P(t_1 t_2 t_3 \ldots t_n)) \tag{4.4}$$

Note that a low perplexity score corresponds to a high probability score. Thus, the lower the perplexity score of a sentence is, the more closely the language model captures the sentence.

One problem of applying N-gram model]in real tasks is that it assigns a zero probability to a sentence if an N-gram in the sentence does not appear in the training corpus. To deal with this problem, many *smoothing* techniques have been proposed in the literature. A smoothing technique assigns a small but non-zero probability to an N-gram that does not appear in the training corpus. One of the well known smoothing technique is the Katz backoff model [43]. It replaces the probability a word w considering the prior $N - 1$ words, with the probability of w considering the prior $M - 1$ words (where $M < N$), if the earlier probability is zero. In effect, it reduces a N-gram model to a M-gram model, where M is less than N, if an N-gram does not exist in the training corpus.

4.4 Proposed Approach

Figure 4.3 shows the overall framework of NIRMAL. The approach includes three major phases: the model creation phase, tweet ranking phase, and evaluation phase. In the model creation phase, NIRMAL learns a language model from StackOverflow data. In the tweet ranking phase, NIRMAL first uses the learned model to compute the perplexity score of each tweet. The lower the perplexity score, the more likely the tweet is software related. NIRMAL then ranks the tweets in ascending order of

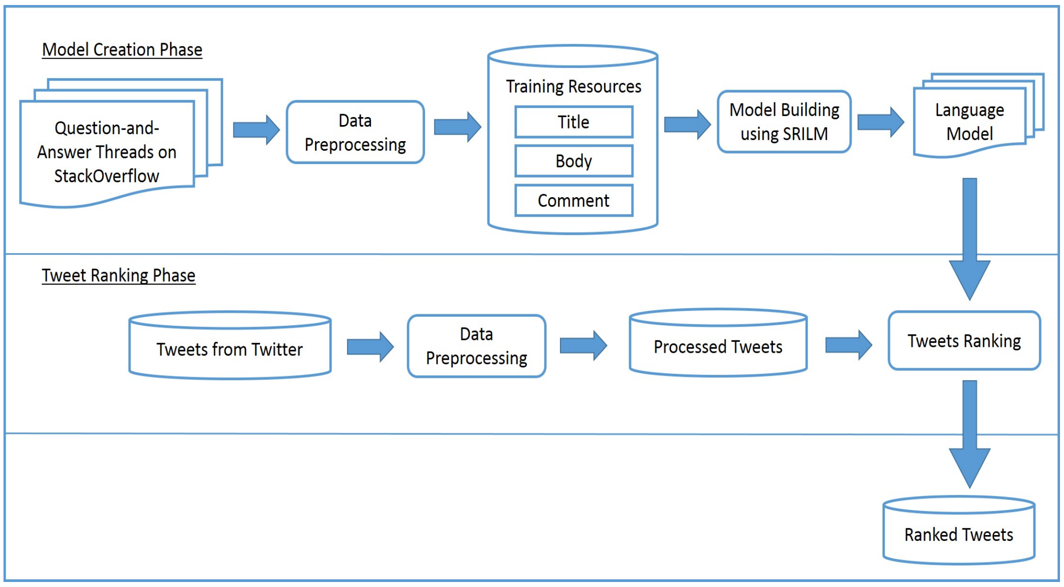


Figure 4.3: Framework of the proposed approach - NIRMAL

their perplexity scores and returns this ranked list. The three phases are described in more detail in the following subsections.

4.4.1 Stack Overflow Data Acquisition & Preprocessing

We used the Stack Overflow data dump that is provided in the following website: `archive.org/download/stackexchange`. In the website, there are many files corresponding to contents from various StackExchange websites (including StackOverflow). We use the following two files: Posts.7z[5] and Comments.7z[6]. Posts.7z contains the title and body (i.e., question and answers) of posts that appear in StackOverflow. Comments.7z contains comments that people give to the questions and answers in StackOverflow. These files contain contents posted in Stack Overflow from July/September 2008 to September 2014. There are a total of 7,990,787 titles, 21,736,594 bodies (i.e., questions + answers), and 32,506,636 comments.

Since there are too many bodies (i.e., questions + answers) and comments, to

[5]https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z
[6]https://archive.org/download/stackexchange/stackoverflow.com-Comments.7z

reduce the time it takes to learn a language model, we only used 8,000,000 of them. We randomly selected 8,000,000 bodies and comments from the data dump. We also performed simple text pre-processing. We removed all punctuation marks and URLs from the sentences in the titles, bodies, and comments. We also changed all words to their lower case.

4.4.2 Language Model Building

We used SRILM [92], a popular language modelling toolkit, to create an N-gram language model. SRILM takes as input a set of documents, a parameter N, and outputs an N-gram language model that characterizes the regularities of text in the input set of documents. SRILM performs smoothing following the Katz backoff model [43]. Thus, it reduces a N-gram model to a M-gram model, where M is less than N, if an N-gram does not exist in the training corpus.

4.4.3 Twitter Data Acquisition & Preprocessing

To collect tweets, we first obtained a set of microbloggers that are more likely to generate software related contents. We started with a collection of 100 seed microbloggers who are well known-software developers[7]. Among these seed microbloggers we have `codinghorror` which is the Twitter alias of Jeff Atwood, the founder of StackOverflow. Next, we analyzed the *follow* links of these microbloggers on March 1, 2013, to identify other microbloggers that follow or are followed by at least 5 seed microbloggers. We added these other microbloggers to the set of seed microbloggers to get a set of 90,883 microbloggers. After, we had identified the target microbloggers, we downloaded tweets that are generated by these microbloggers from April 1 to April 31, 2013. We downloaded these tweets using the Twitter REST API. We have a Twitter whitelist account that allows us to make 20,000 API calls every hour. In total, we collected 6,294,015 tweets.

[7]http://www.noop.nl/2009/02/twitter-top-100-for-softwaredevelopers.html

We performed simple pre-processing on the collected tweets. We removed punctuation marks and URLs, and also changed all words into their lowercase.

4.4.4 Ranking of Tweets

To rank tweets, we made use of two sources of information: first, we used the perplexity score that is output by the language model; second, we computed a scaling factor based on the repetitiveness of words in a tweet. We found that many tweets with repetitive contents, e.g., “1 1 1 1 1 1 1”, “a a a a a a”, are rather meaningless. Most meaningful tweets do not have a high number of repetition. We computed the scaling factor of a sentence S using the following equation:

$$S_t(S) = \frac{wc_t(S)}{wc_u(S)} \tag{4.5}$$

In the above equation, $wc_t(S)$ is the number of words in the sentence S and $wc_u(S)$ is the number of unique words in the sentence S. Note that the lower the scaling factor the less repetitive a tweet is and the more likely it is meaningful.

Given a sentence S after we have computed its perplexity score (i.e., $PP(S)$) and its scaling factor (i.e., $S_t(S)$), we can compute its revised perplexity score, denoted by $PP^R(S)$, as follows:

$$PP^R(S) = PP(S) \times S_t(S) \tag{4.6}$$

The lower the ranking score of a tweet the higher is the likelihood of it to be software related and not meaningless. Note that due to smoothing using Katz backoff model, although both “1 1 1 1 1 1 1”, “a a a a a a” do not appear in the StackOverflow data, SRILM will assign a relatively low perplexity score to both sentences since “1” and “a” appears often in the StackOverflow data. Thus, we need to leverage the repetitiveness of contents in a tweet to increase the score of these meaningless tweets.

4.5 Experiments & Analysis

In this section, we describe our dataset and experimental settings, followed by our evaluation metric. We then present our four research questions and the results of our experiments.

4.5.1 Dataset and Settings

The detailed statistics of the Twitter and StackOverflow datasets that we use in this experiment are shown in Table 4.1. The number of words in the tweets that we collected amounts to more than 77 million. The number of words in the titles, bodies (i.e., questions + answers), and comments that we collected amounts to, slightly more than 39 thousand, 725 million and 200 million, respectively.

Table 4.1: Statistics of Twitter Data and Stack Overflow Data.

Corpus	#Documents	#Words
Twitter	6,294,015	77,491,505
StackOverflow (Title)	7,990,787	39,786
StackOverflow (Body)	8,000,000	725,449,601
StackOverflow (Comment)	8,000,000	200,584,369

NIRMAL accepts two inputs: the dataset used to learn a language model and the parameter N of N-gram. By default, unless otherwise stated, we use the StackOverflow (Title) corpus (i.e., the titles of the StackOverflow posts) to learn a language model, and set the value of N to 4. We run the experiment using the following machines: Preprocessing and tweet ranking steps are run on Intel Core i5-4570 3.2 GHz CPU, 8 GB RAM desktop running Windows 7 64 bit. All SRILM related steps are performed on a 7 core Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz, 64 GB RAM server running CentOS release 6.5.

4.5.2 Evaluation Metrics

We use NIRMAL to sort the 6.2 million tweets and we manually inspect the top-K tweets that are returned by NIRMAL. We evaluate the effectiveness of our approach using accuracy@K. accuracy@K is defined as the proportion of tweets in the top-K positions that are software related. accuracy@K has also been used to evaluate other past studies [38, 130, 136].

4.5.3 Research Questions

Our experiments aim to answer following four research questions that assess the strengths and limitations of NIRMAL and several baseline approaches.

RQ1: How effective is our approach in identifying software related tweets?

In this question, we want to evaluate how effective is NIRMAL in ranking tweets such that the software related ones are ranked higher than the non software related ones. To answer this research question we simply run NIRMAL with the default setting on the 6.2 million tweets and manually evaluate the top-K tweets that are returned by NIRMAL. We report the accuracy@K scores that are achieved by NIRMAL. We compare the performance of NIRMAL with the performance of a random model that randomly labels K tweets as software related.

RQ2: What are the effects of varying NIRMAL inputs on its effectiveness?

NIRMAL accepts two kinds of inputs: the value N for the N-gram model, and the dataset used to train the N-gram model. In this research question, we want to investigate the impact of using different values of N and different datasets on the overall effectiveness (i.e., accuracy@K scores) of NIRMAL. We investigate four different N values, i.e., 1,2,3, and 4, and five different datasets: StackOverflow (Title), StackOverflow (Body), StackOverflow (Comment), StackOverflow (Title) $\bigcup$ StackOverflow (Body), StackOverflow (Title) $\bigcup$ StackOverflow (Body) $\bigcup$ StackOverflow (Comment).

RQ3: How efficient is our approach?

Many new tweets are continuously generated every second. For our approach to work in practice, it needs to be able to process new tweets efficiently. In this research question, we investigated the time it takes for NIRMAL to learn a language model and the time it takes to compute the revised perplexity score of a tweet. Since a trained language model can be used to label many tweets before it needs to be retrained, the time NIRMAL takes to learn a language model can be long (e.g., a few hours) but cannot be excessively long (e.g., a few months). On the other hand, the time NIRMAL takes to compute the revised perplexity score of a tweet needs to be very short (i.e., less than a second).

RQ4: Could our approach improve the effectiveness of the keyword based approach?

Achananuparp et al. and Tian et al. have used a set of keywords to detect if a tweet is software related or not. However, many tweets that contain one or more of the keywords are not software related. In this research question, we investigated whether we can use NIRMAL to effectively sort tweets that have been filtered such that the software related ones appear in the top of the list. To answer this research question, we first filtered the 6.2 million tweets using the 100 keywords that Achananuparp et al. and Tian et al. used. In total, among the 6.2 million tweets, we have 227,225 tweets that contain at least one of the keywords.We then selected a random sample of 200 tweets and calculated the accuracy@K scores for *keyword only* approach. We then applied NIRMAL to sort all the 227,225 keyword containing tweets and manually evaluated the top-K tweets to compute the accuracy@K score to check if NIRMAL is able to improve the accuracy of keyword based approach.

4.5.4 Research Results

In this section, we present our experiment results that answer each of the research questions raised in the previous section.

RQ1: Effectiveness of Our Approach

The results of our experiment are shown in Table 4.2. From the results we can note that the accuracy@K of NIRMAL ranges from 0.695 to 0.900 using the default setting. When we investigated the top-10 tweets, we found that 90% of them are software related. When we investigated the top-200 tweets, we found that 69.5% of them are software related when NIRMAL is used. On the other hand for a random model only 24% of the top-200 tweets were software related. This shows that NIRMAL is accurate, and also that the tweets ranked higher in the list are more likely to be software related than those ranked lower in the list.

Table 4.2: acc@K (i.e., accuracy@K) of NIRMAL for Various K

Approach	**acc@10**	**acc@50**	**acc@100**	**acc@150**	**acc@200**
NIRMAL	0.900	0.820	0.720	0.707	0.695
Random	0.400	0.280	0.280	0.220	0.240

RQ2: Effectiveness of Various Parameter Settings and Learning Resources

Varying the parameter N of the N-gram model. The results of our experiment are shown in Table 4.3. From the results we note that if we increase N for N-gram language model the accuracy@K increases for all values of K, e.g., The accuracy@200 increases from 0.120 to 0.695 as we move from 1-gram to 4-gram model.

Table 4.3: Effect of Varying N on the Performance of NIRMAL

N	**acc@10**	**acc@50**	**acc@100**	**acc@150**	**acc@200**
1	0.000	0.140	0.140	0.127	0.120
2	0.500	0.460	0.460	0.473	0.485
3	0.600	0.640	0.680	0.660	0.630
4	0.900	0.820	0.720	0.707	0.695

Varying the training corpus. The results of our experiment are shown in Table 4.4. From the results we note that for any N-gram language model the highest values of accuracy@K were achieved when the training corpus containing only Titles was used. This can be explained as the Titles will generally contain less noise, i.e., natural language text not related to software. However the Body and Comments contain a lot of normal language text, as well as code samples and fragments, which should explain the relatively lower accuracy scores attained by language models created using their corpus.

Table 4.4: Effect of Using Different Training Corpus on the Performance of NIRMAL. T = Title, B = Body, C = Comment, TB = Title + Body, TBC = Title + Body + Comment.

Corpus	acc@10	acc@50	acc@100	acc@150	acc@200
T	0.900	0.820	0.720	0.707	0.695
B	0.300	0.560	0.530	0.500	0.475
C	0.500	0.380	0.280	0.227	0.200
TB	0.400	0.640	0.560	0.540	0.500
TBC	0.400	0.600	0.540	0.447	0.435

RQ3: Efficiency of Our Approach

The results of our experiment is shown in Table 4.5. We show the time NIRMAL takes to create a language model from the StackOverflow title data (i.e., Model Creation Time), and the average time NIRMAL takes to compute the revised perplexity score of a tweet (i.e., Model Compu. Time), for various values of the N parameter. All the times are shown in seconds. We can observe that as N increases the time to create a model also increases. This is pretty evident because the model will need to consider a higher number of N-grams (word pairs) when N increases. However change in N seems to have a negligible effect on the time required to calculate revised perplexity score for a new tweet. Please note that perplexity score calculation time has been averaged over score calculation time for all 6,294,015 tweets.

Table 4.5: Efficiency of NIRMAL

N	Model Creation Time (in Sec.)	Score Compu. Time (in Sec.)
1-gram	14	0.000268827
2-gram	46	0.000261518
3-gram	101	0.000261359
4-gram	175	0.000278042

RQ4: Integration with Keyword Based Method

The experiment results are shown in Table 4.6. We can clearly observe that applying the NIRMAL to the keyword approach improves the accuracy@K for all values of K. Our results show that NIRMAL can be used to improve the accuracy score up to 31%. The lowest observed increase of about 11.11% for accuracy@10 value. But this should be seen as a positive result as it was the maximum increase possible at K=10. The new value achieved i.e., 1 (obtained after applying NIRMAL) is the highest value possible value for the parameter accuracy@K. Thus, we can deduce that applying NIRMAL to a keyword approach seems to result in an improved performance.

Table 4.6: Keyword VS. NIRMAL + Keyword

Approach	**acc@10**	**acc@50**	**acc@100**	**acc@150**	**acc@200**
Key.	0.900	0.700	0.740	0.753	0.755
NIRMAL + Key.	1.000	0.920	0.950	0.967	0.975

4.5.5 Threats to Validity

In this section, we discuss threats to three types of validity, i.e., internal, external, and construct validity.

Threats to internal validity. Threats to internal validity relate to errors in our experiments and our labelling. Most of our experimental process is based on SRILM, a commonly used language model learning and application tool. We believe the code of SRILM is stable and reliable. To label the tweets as software related or not, we asked one PhD student with more than 5 years of experience in software industry and more than 10 years of experience in programming to manually label the tweets. We believe the PhD student has enough expertise to decide if a tweet is software related or not. When labeling the tweets, the PhD student not only reads the tweets but also opens the URLs contained in the tweets (if needed). The labeling process might be subjective, however, since one person labels all tweets the judging criteria used remains consistent.

Threats to external validity. Threats to external validity relates to the generalizability of our approach and evaluation. In this work, to reduce threats brought by using a small training corpus, we have downloaded and used millions of titles, questions, answers, and comments from the official StackOverflow dump which contains contents posted in Stack Overflow from 2008 to 2014. We have used NIRMAL to rank more than 6.2 million tweets that are generated by more than 90 thousand microbloggers over a two month period. We have manually labeled the top-200 tweets generated by NIRMAL. In the future, to reduce the threats to external validity, we plan to use NIRMAL to rank a larger number of tweets generated by more microbloggers. We also plan to manually label a larger number of tweets.

Threats to construct validity. Threats to construct validity relates to the suitability of our evaluation metric. In this work, we use accuracy@K to measure the effectiveness of our approach. This metric is intuitive and it has been used in many previous studies, e.g., [38, 130, 136]. Thus, we believe there is little threat to construct validity.

4.6 Conclusion

Twitter has become a popular means to share and disseminate information. To date, there are hundreds of millions of Twitter users generating billions of microblogs (aka tweets). Software developers are also using Twitter, even for their professional activities. Singer et al. found that software developers use Twitter to get awareness of people and trends, extend their technical knowledge, and build connections with other developers [89]. Unfortunately, developers often find it a challenge to deal with the many irrelevant tweets (i.e., noises) in their Twitter streams. Many developers follow many people that generate many tweets (many of which are irrelevant) that get broadcasted to them every day.

To make Twitter a better tool for developers in their professional activities, we propose a new approach that can help developers identify software related tweets from the mass of other irrelevant tweets. Our approach, named NIRMAL, trains a language model from a corpus of software related contents on Stack Overflow. The trained language model infers the regularities of software related contents and use these regularities to compute the likelihood of a tweet to be software related. To improve the performance further, NIRMAL also considers the repetitiveness of words in a tweet that can be used to differentiate between informative and meaningless tweets. In our experiment, we have used NIRMAL to rank a set of 6.2 million tweets generated by more than 90 thousands microbloggers. Most of the tweets are not software related while only a minority of them are software related. The experiment results show that NIRMAL can achieve an accuracy@200 score of up to 0.695 which is greater than the accuracy@200 score of a random model by up to 192%. Furthermore, NIRMAL can be used to improve the accuracy score of a keyword based approach by up to 31%.

As a future work, we plan to build N-grams with larger N and evaluate how they perform w.r.t parameters of accuracy and computational performance. We plan to investigate the effect on performance of current models by adding more preprocessing steps such as stemming and stop word removal. We also plan to propose an approach that can summarize the identified software related tweets to help developers better manage the large number of tweets that they receive daily.

Chapter 5

Mining Informative Online Resources Shared by Developers on Social Media

5.1 Introduction

Software development is a field which evolves rapidly, so software developers always need to keep themselves up to date with new knowledge and methodologies. Learning continuously and serendipitously may help them to solve new, unseen and/or complex challenges that they may encounter during their software development tasks. Storey et al. found that *keeping up with new technologies* is a major challenge faced by software developers today [96]. They also found that developers use media such as Twitter to keep them up to date with the latest trends and to extend their software knowledge [89].

In this work, we present an approach to support the serendipitous learning of developers by harnessing Twitter as a knowledge repository. Past research has shown that Twitter is used by software developers to share important information with other fellow developers [89, 11, 103]. Sharing links in the form of URLs (Uniform Resource Locators) of various software related articles and multimedia is a popular activity in software engineering Twitter space [86]. Twitter has been found to be better at serendipitously exposing developers to latest updates and developments in technology when compared to search engines [89]. Also, consideration of the URLs on Twitter allows us to reduce the search space for finding popular and relevant URLs, and also to infer the social approval of links shared. Unfortunately, even on Twitter, finding URLs to relevant and useful articles for a particular domain of interest (e.g., Java) is not an easy task. Developers need to identify many relevant Twitter users to follow, and sieve through a large amount of tweets that they may generate, which often result in information overload. These challenges have been validated by Singer et al. in their survey with developers [89].

To address the above mentioned challenges, we propose an unsupervised and a supervised approach to harvest and rank URLs linked to contents that are popular and relevant to a particular domain of interest from Twitter. Both output a sorted list of URLs sorted based on their likelihood to be popular and relevant to the domain of interest, where domain is characterized by a set of keywords (e.g., {"Java"}). The supervised approach also requires as an input a small training set, which contains URLs that are manually assigned with relevance ratings ranging from 0 (highly irrelevant) to 3 (highly relevant). Both of the two approaches characterize a URL in terms of 14 features that are grouped into three families: content features, popularity features, and network features. Our unsupervised approach makes use of Borda count [4], a popular data fusion technique, to rank URLs based on their features. Our supervised approach makes use of *Learning to Rank* [51], a popular information retrieval technique, to build a ranking model from the labeled URLs, which can then be applied to rank a set of URLs based on their likelihood to be informative.

In this preliminary study, we evaluate the two proposed approaches on a dataset of 577 unique URLs found among 2,104 tweets posted by people potentially interested in software development. These 2,104 tweets were filtered from about 3,980,397 tweets posted in November 2015 based on the condition that they contain the keyword "Java". We measure the effectiveness of our approaches in ranking these URLs in terms of Normalized Discounted Cumulative Gain (NDCG) [39]. NDCG scores are computed based on the relevance ranks of the URLs which were manually labeled by two study participants. The participants label the data independently and then resolve their differences in order to create the final ground truth. The URLs in our ground truth data have been assigned relevance ratings in the range 0 (highly irrelevant) to 3 (highly relevant). NDCG was designed for settings such as in our work where the relevance ratings are non-binary [39]. NDCG score is between 0 and 1, with 1 indicating an ideal ranking algorithm. The experiments show that our proposed unsupervised and supervised approaches can achieve a high NDCG score of 0.719 and 0.832 respectively.

The contributions of this work are as follows:

1. We propose an unsupervised and supervised approach to support developer serendipitous learning using Twitter by ranking URLs to online resources. To the best of our knowledge, no prior study has helped developers in this task. Our approaches sieve through a large number of tweets to automatically extract and rank URLs relevant to a particular domain of interest. Our preliminary evaluation shows that they can achieve reasonably high Normalized Discounted Cumulative Gain (NDCG) scores on a dataset of 577 URLs related to the keyword 'Java'.

2. We propose 14 features from three categories, i.e., content features, popularity features, and network features, to comprehensively characterize a URL given a set of keywords describing a domain of interest.

The structure of the remainder of this work is as follows. In Section 5.2, we describe our proposed approach that extracts and ranks informative URLs from Twitter. In Section 5.3, we present our experiment settings and results. We finally conclude and mention future work in Section 5.4.

5.2 Approach

Our approach has four steps, i.e., Data Acquisition, Feature Extraction, Unsupervised Recommendation and Supervised Recommendation, as shown in Figure 5.1.

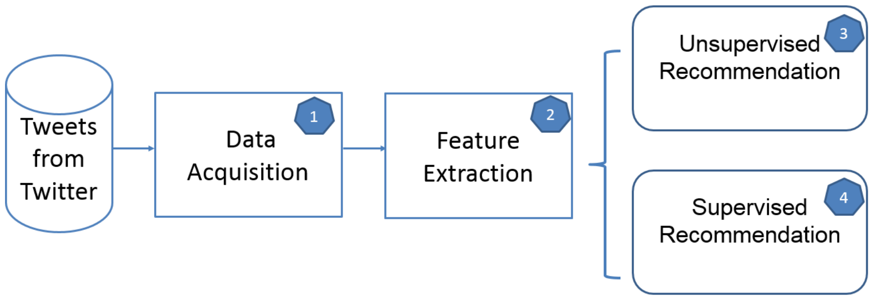


Figure 5.1: Approach Overview

5.2.1 Data Acquisition

We first identify some users on Twitter who are potentially interested in software development. We start with a set of well known software developers who are also present on Twitter. We then process the profile of these seed users to find all the other users who follow or are followed by at least n of these seed users. The approach has been used in several previous works [104, 85, 86]. We then download and process the tweets of these identified Twitter users on a period of time, filtering tweets using keywords that characterize a domain of interest. Next, we extract URLs shared in these tweets. These URLs are typically shortened by Twitter itself or by users using a URL shortening service, e.g., `https://goo.gl/`. If a URL has been shortened by Twitter, it maintains a reference of the expanded URL in the tweet's meta data. In case the Twitter user had used an external service to shorten the URL, we use a browser to expand the short URLs to their expanded forms. Then we remove the duplicates among expanded URLs. We also remove URLs which correspond to broken links and error pages. In the end, we have a set of valid URLs along with the other associated information such as tweet content and user data.

5.2.2 Feature Extraction

which help us to find useful URLs w.r.t. a particular domain of interest represented by a set of keywords. We have categorized the features into three broad categories: *Content Features*, *Popularity Features*, and *Network Features*. We briefly explain the features for each category below.

- ***Content Features.*** These features are based on the similarity between the input keywords, which characterize the domain of interest, and various textual contents that are linked to a URL. These textual contents can come from various sources including the tweet mentioning the URL, the text of the webpage pointed to by the URL, and the text contained in the profile of the user sharing the tweet containing the URL. We consider the set of keywords as a document, and the various textual contents as documents too.

  - **CosSimT:** This feature corresponds to the cosine similarity between the keywords related to our domain of interest and the combined text of all the tweets which mention a particular URL. Generally, when users share an important URL on Twitter, they give a brief description of the URL they are sharing, so as to ensure that their subscribers have an idea about the importance as well as the background of the URL being shared. Thus the text in the tweets can serve as an important pointer to assess the relevance of the URL being shared in the tweets.

    For each URL, we collate the text of all the tweets that mentioned the URL in one document and calculate the cosine similarity score of the keywords with the collated document. The resultant score is the value of the *CosSimT* feature. Intuitively, this score represents the degree of relevance between the tweets which mentioned the URL and the keywords (representing the domain) being queried. A higher score indicates that the URL is more relevant to the domain of interest.

  - **CosSimW**: Through this feature, we measure the cosine similarity score between the keywords and the text contents on the webpage which a URL link resolves to. The information present on the webpage is an important input in defining the relevance of the page w.r.t. the keywords. For calculating the value of this feature, we download each URL's web data as text. We then compute the similarity score between the keywords and the text document consisting of text extracted from the webpage of the URL. A high score for *CosSimW* suggests that the URL is highly relevant to the domain of interest.

  - **CosSimP**: This feature measures the cosine similarity between the input keywords and the combined text from all the profile data of users who posted a particular URL. URLs shared by software developers often relate to a particular software domain that the developers are interested in. Generally, Twitter users provide some information about their interests and jobs in the profile section of their Twitter page. To compute the value of this feature, we combine the profile data of all the users who posted an URL in one document, and then calculate the similarity score between the keywords and the document. If the *CosSimP* score of a URL is high, it suggests that users who posted this URL are likely to be knowledgable in the domain of interest.

- ***Popularity Features.*** These features measure the popularity of a URL Link. Many people share URLs that they find informative on Twitter. Others help to broadcast these URLs by retweeting those they think are informative and relevant to their domain of interest, and which also might be helpful to other people in their network. Intuitively, URLs that are shared by many are likely to be highly informative. URL popularity can be measured in various ways, e.g., number of tweets mentioning an URL, number of users who mention an

URL, etc.We consider 4 features to measure the popularity of an URL which are briefly described below.

– **NumOfT**: This feature counts the number of tweets or retweets generated by a community of software enthusiasts on Twitter (i.e., users tracked in the data acquisition step) which contain a particular URL. A higher score for this feature means the URL has been shared widely in the software community and thus should be more popular as compared to other URLs which have not been shared much. This feature score serves as the most basic and intuitive way of measuring the popularity of an URL.

– **NumOfU**: This feature counts the number of unique users in a community of software enthusiasts who have shared a particular URL in their tweets. This feature differs from *NumOfT* feature as a user may post the same URL link in multiple tweets. For calculation of *NumOfU* we only consider a user once.

– **NumOfRT**: This feature counts the sum of the retweet counts of all the original tweets that contain the URL link. Retweeting is a feature on Twitter through which a user can broadcast to their followers a tweet published by somebody else. Most users express their liking for a particular tweet by retweeting it. For computing this feature we identify all the original tweets that were retweeted and contain the URL. Then the sum of retweets of all such original tweets constitutes the value of this feature. For this feature, the contribution for the retweet count can come from any user in Twitter network and is not limited to users in our dataset. Thus through *NumOfRT* we try to infer the global popularity of the URL.

– **NumOfF**: This feature counts the sum of the favourite counts of all the tweets and retweets that contain the URL link. For computing this feature we first identify all tweets and retweets containing the URL in our dataset. Next, we compute the sum of the favourite counts of these tweets and retweets. A higher value of this metric also suggests high popularity of the URL.

- ***Network Features.*** We take the network of all Twitter users in our dataset who have posted at least a tweet containing the domain related URL and then infer the network importance of each user present, considering each user as a network node. To measure the importance of user, we use popular centrality metrics proposed in web and social network mining communities [132, 12, 67, 27]. We compute the features by using Jung (`http://jung.sourceforge.net/`). We provide a brief description below. (For a complete description please refer [131]).

  - **Barycenter Centrality**: This feature is computed by taking the reciprocal of the sum of shortest distance of a node to each other node in a network. The barycenter centrality of a Twitter user u is computed as:

    $$BaryC(u) = \frac{1}{\sum_{v \neq u} sdist(u, v)}$$

    In the equation, $sdist(u, v)$ calculates the shortest distance from user u to user v.

  - **Betweenness Centrality**: This feature counts the number of shortest paths from all nodes to all others that pass through a node. The betweenness centrality of a Twitter user u is computed as follows:

    $$BetweenC(u) = \sum_{a \neq b \neq u} \frac{spath(a, b, u)}{spath(a, b)}$$

    In the equation, $spath(a, b, v)$ computes the number of shortest paths between user a and user b that pass through user u. Similarly, $spath(a, b)$ computes the number of shortest paths between user a and user b.

  - **Closeness Centrality**: This feature is computed by taking the reciprocal of the average shortest distance of a node to all the other nodes in a network. The closeness centrality of a Twitter user u is defined as follows:

$$CC(u) = \frac{n-1}{\sum_{v \neq u} sdist(u,v)}$$

In the equation, n is the total number of users in the network. $sdist(u,v)$ computes the shortest distance from user u to user v.

– **Eigenvector Centrality**: This feature measures the importance of a node based on the importance of its neighboring nodes. The values of eigenvector centrality for nodes in the network is computed as follows:

$$\alpha(I - \beta R)^{-1} R1$$

In the above equation, α is a scaling vector for normalizing the score, I is the identity matrix, R is the adjacency matrix representing the network, β is the weighting factor for the adjacency matrix, and $R1$ is a matrix where the contents of all its cells are ones. Since the value of this metric is often very small, in this work we compute the reciprocal of this metric. We use the default values of α and β in Jung.

– **Hubs and Authorities**: Hubs and Authorities are two scores to measure node importance in network. They are computed based on the Hyperlink-Induced Topic Search (HITS) algorithm proposed by Kleinberg [46]. These two scores of a Twitter user u are computed as follows:

$$Hub(u) = \sum_{i=1}^{n} Auth(u),$$

$$Auth(u) = \sum_{i=1}^{n} Hub(u)$$

In the equation, n is the total number of users in a network, $Hub(u)$ computes the hub score for node u, and $Auth(u)$ computes the authority score for node u.

– **PageRank**: PageRank (PR) is a node importance measurement proposed by Brin and Page [67]. The PR algorithm computes a probability to represent the likelihood of a particular node being visited while randomly traversing edges. The PR algorithm runs iteratively. At a iteration i, the PR score of a Twitter u is defined as follows:

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

In the equation, d is the probability that a random walker continues to visit other users (aka the *damping factor*), N is the number of users in the network, $B(u)$ refers to the set of users that link to u, and $L(v)$ is the set of users that v links to. The iteration stops when PR score of each user converge.

5.2.3 Unsupervised Recommendation

Based on the 14 feature scores, we use Borda Count [4] to arrive at a combined score for a URL and then rank the URLs based on this combined score.

Borda Count works by first assigning a *rank* for each feature score to a URL. For each feature score, we create a list of all URLs that we have harvested in the data acquisition step, and sort them in descending order of their feature scores. The *rank* of a URL for a feature is then defined as the position of the URL in the sorted list. Next, for each feature score, after we have the rank of a URL, we can compute its *ranking point*. It is calculated by subtracting the *rank* of the URL from the total number of URLs. After we have the *ranks* and *ranking points* for all URLs and features, we can compute the URL's combined score. Let u_i denotes the i^{th} URL and $rp_j(u_i)$ denotes the ranking point assigned to u_i for the j^{th} feature. Also, let N_f

denotes the number of feature scores per URL and N_u denotes the total number of URLs in our data set. The combined score of a URL u_i can be calculated as follows:

$$BordaScore(u_i) = \frac{\sum_{j=1}^{N_u}(rp_j(u_i))}{N_f \times N_u}$$

In the above equation, the combined score is the summation of all the *ranking points* divided by the product of N_f and N_u. After obtaining the combined score, we rank the URLs in the descending order of their combined scores. The URL having the highest combined feature score is considered the most relevant, and the URL having the lowest score is considered as the most irrelevant.

5.2.4 Supervised Recommendation

We use *Learning to Rank* [51] approach to train a supervised model which is then used to assign ranks to URLs. In the learning phase of our supervised approach, we consider a set of URLs as training data and based on the feature scores of these URLs and their corresponding manually assigned labels, we learn a ranking function $f(u)$. This function $f(u)$ can be considered as the weighted sum of all the features of a URL u, and during the learning phase it tries to learn these weights or parameters of the features through optimization. This ranking function when applied to unseen test URLs (also represented as their corresponding feature vectors) assigns scores to the URLs. Based on the scores provided by $f(u)$, all the test URLs can be ranked in the descending order. This sorted list is considered as the recommended result. In this work, we make use of a popular off-the-shelf implementation of a learning to rank algorithm, SVM^{rank}, which is made available from `https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`.

5.3 Experiments and Results

In this section, we first present the process of creating ground truth set. Next, we describe our experiment setting and evaluation metric. Finally, we present our research questions and the results of our experiments which answer the questions.

5.3.1 Dataset

In the data acquisition step, as the seed set of Twitter users, we use a list of top 100 popular developers on Twitter given in: `http://noop.nl/2009/02/twitter-top-100-for-software-developers.html`. We set n as 5 (i.e., we find all other users who follow or are followed by at least 5 of these seed users). Moreover, we collect tweets made on November 2015, and filter tweets using keyword "*Java*". We are able to extract 2,104 of such tweets and 577 unique and valid URLs along with their associated information. More URLs could be gathered if we expand our Twitter user base and the period of time the tweets were made. We leave the gathering of an extended dataset for a more comprehensive evaluation as future work.

Next, we manually assign relevance score labels on a scale of 0 to 3 for each of the selected 577 unique URLs we extracted. The data is labeled by 2 persons, both having more than 4 years of professional programming experience in Java. The labellers are provided with the 577 URLs and asked to browse the websites pointed to by the URLs and then have to assign a score to the URL, with a score of 3 being assigned if the content linked with the URL is highly relevant and shareable, 2 being assigned if the content is relevant but not worth sharing, 1 being assigned if URL content was marginally relevant and not shareable, and 0 being assigned if the content is highly irrelevant. For the URLs where the two labellers have a disagreement, they have to sit down together to discuss and agree to a final label. Table 5.1 shows the distribution of the labels for the 577 URLs.

Table 5.1: Distribution of Scores for the 577 URLs

Label Assigned	0	1	2	3	Total
#URLs	115	77	184	201	577

5.3.2 Experiment Setting and Evaluation Metrics

By default, we perform 10-fold cross validation to investigate the effectiveness of our approach. As an evaluation metric, we make use of Normalized Discounted Cumulative Gain (NDCG). NDCG measures the performance of a recommendation system by evaluating its capability to recommend more relevant URLs as the top results and less relevant ones as the bottom results. NDCG gives a score between 0 and 1 to the recommender system it evaluates. The closer the NDCG score of a system is to 1, the more effective it is at recommending informative URLs. We use the following formula to calculate NDCG:

$$NDCG = \frac{DCG - WDCG}{IDCG - WDCG}$$

In the above formula, DCG is a Discounted Cumulative Gain score [51] of the URL relevance, IDCG is the ideal DCG score (i.e., informative URLs are listed before less informative ones), and WDCG is the worst DCG score (i.e., all less informative URLs are listed before more informative ones). The following equation is used to compute DCG, where rel_i is the rating assessment for the URL at position i in the ranking:

$$DCG = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log(i)}$$

The main concept of DCG is that relevant documents (in our case, relevant URLs) appearing lower in a search result list corresponds to a poorer result.

5.3.3 Research Questions

RQ1: How effective are our unsupervised and supervised approaches in recommending informative URLs?

In this research question, we investigate the effectiveness of our two approaches based on the NDCG metric. Table 5.2 shows the NDCG scores of our approaches. The NDCG score for the unsupervised approach is 0.719 while that for the supervised approach is 0.832. The supervised approach can outperform the unsupervised one by 15.71%. Table 5.3 shows some examples of URLs that are recommended by our approach.

Table 5.2: NDCG Scores of Our Proposed Approaches

Approach	NDCG Score
Unsupervised	0.719
Supervised	0.832

Table 5.3: Some Examples of Recommended URLs

URL
`www.infoq.com/articles/Java-The-Missing-Features`
`http://github.com/zeroturnaround/java-fundamentals`
`www.adam-bien.com/roller/abien/entry/java_8_ infinite_stream_of`

RQ2: How sensitive is our supervised approach on the amount of training data?

In this research question, we investigate the impact of reducing the amount of training data on the effectiveness of our supervised approach by performing k-fold cross validation and varying the value of k from 2 to 10. From Table 5.4, we can see that the performance of our supervised approach remains stable across various values of k, and is not overly sensitive.

Table 5.4: NDCG Scores for Different k

k	**NDCG**	**k**	**NDCG**	**k**	**NDCG**
10	0.832	7	0.845	4	0.837
9	0.825	6	0.834	3	0.847
8	0.833	5	0.842	2	0.843

5.3.4 Threats to Validity

Threats to *internal validity* refer to experimenter biases. We have tried to mitigate this threat by asking two persons to independently rate the relevance of the webpages pointed to by the URLs, and later meet to resolve their disagreements. Threats to *external validity* refer to the generalizability of our findings. For this preliminary work, we have considered one domain of interest, namely Java programming language, using the keyword "Java" to characterize this domain. In the future, we plan to reduce this threat further by considering other domains and/or keywords in addition to Java domain. Threats to *construct validity* correspond to the suitability of our evaluation metric. In this work, we make use of Normalized Discounted Cumulative Gain (NDCG) which is a standard information retrieval metric and has also been used in many past software engineering studies, e.g., [80, 29]. Therefore, we believe that threat to construct validity is minimal.

5.4 Conclusion and Future Work

Software developers using channels such as Twitter serendipitously learn about new methodologies and keep their skills and knowledge up to date. Unfortunately, given the huge number of choices developers have at their disposal, identifying which resources and channels to follow and what to ignore is a major challenge for them [96].

We propose two approaches, one unsupervised and one supervised, to search and rank URLs harvested from Twitter which can support developers in their serendipitous learning tasks. These approaches are based on 14 features which characterize a URL's relevance and informativeness from three dimensions: 1) *content* features which capture similarity of the input domain specific keyword with the textual contents of tweets, webpages pointed to by the URLs, and user profiles, 2) *popularity* features which characterize the popularity of the tweets containing the URL on Twitter, 3) *network* features which characterize the importance of the user posting the URL on Twitter. In our preliminary experiments, we evaluate the two approaches on a set of 577 URLs. The experiments show that our unsupervised and supervised approaches can achieve a reasonably high Normalized Discounted Cumulative Gain (NDCG) score of 0.719 and 0.832 respectively.

As a future work, we plan to improve the effectiveness of our approach further by the incorporation of additional features and the design of more sophisticated algorithms. We would also like to enlarge the scale of our experiments to consider more tweets collected over a longer period of time and also to add more channels to mine URLs. Moreover, we plan to build a site that shares URLs of informative resources that are harvested by our proposed approach and gets continuously updated in real time.

Chapter 6

Recommending Experts in the Software Engineering Twitter Space

6.1 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 [114]. Twitter allows users to post short messages that are broadcasted to other users who have chosen to *follow* them. These messages can be further *retweeted* (i.e., propagated) to reach even a 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[48].

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" [89]. 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 6.2). 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. which observed that many developers follow thought leaders from their technological niches [89].

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: he/she must be an experienced software developer in a specialized domain, and he/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 that 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 one 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 [68], as well as Klout [77], 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 following research questions which have been discussed in detail in Section 6.5:

- **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 non-gurus?
- **RQ4:** Which feature values have the best predictive power across each domain?
- **RQ5:** What is the cross domain performance of our approach?

6.2 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 do software developers like to follow. To understand this, we designed an open ended survey. Our survey consisted of three key questions:

- The first question asks whether a respondent develops software systems and uses Twitter in his/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 their years of experience as a software developer (less than 5 years, 5-10 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 [82, 21]. Specifically, we used open card sort [37] in order to develop categories of Twitter accounts that software developers like to follow. Two PhD students were 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 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 3 respondents into a special category *Others*. The open card sort process was performed together by two people, one of them being the author of this dissertation and other being a PhD student in computer science. Our process is similar to negotiated agreement technique described in [15]. As the card sorting has been performed together, there is no inter-rater agreement number. Many previous studies involving card sorting have also followed a similar process [52, 90, 45, 44, 3].

Survey Participants: We targeted software developers who are present on Twitter. Following [1, 104, 85, 86], we collect a set of 161,067 Twitter users who are potentially interested in software development – see Section 6.5.1 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[1]. 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 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 author of this dissertation has send personalized Twitter direct messages to hundreds of these users, requesting them to fill the survey. In total we have contacted 213 developers, out of which 38 developers responded back by filling 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.

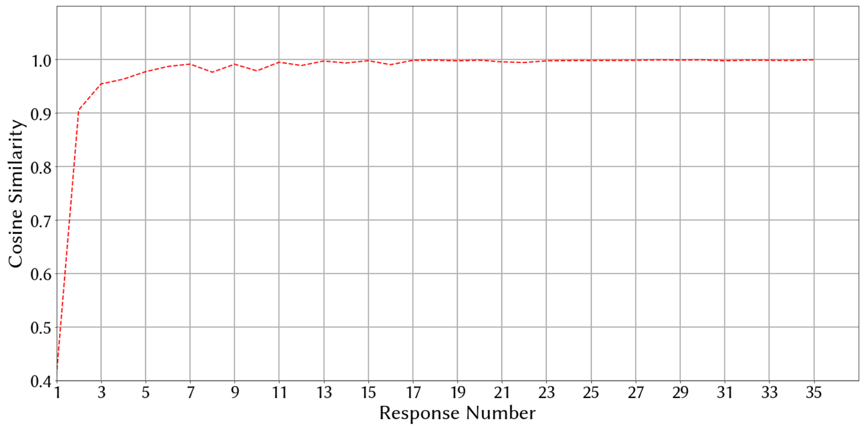


Figure 6.1: Graph showing saturation of $CosSim_n$ score

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 [98], sampling for a survey can be terminated when collecting new data does not generate any new information. During the survey we observed that after we

[1]`https://support.twitter.com/articles/14606`

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 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 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 [58]. 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 6.1. We can note that $CosSim_n$ stabilizes after the 23^{rd} 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.

Table 6.1: Categories of Twitter 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 which 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

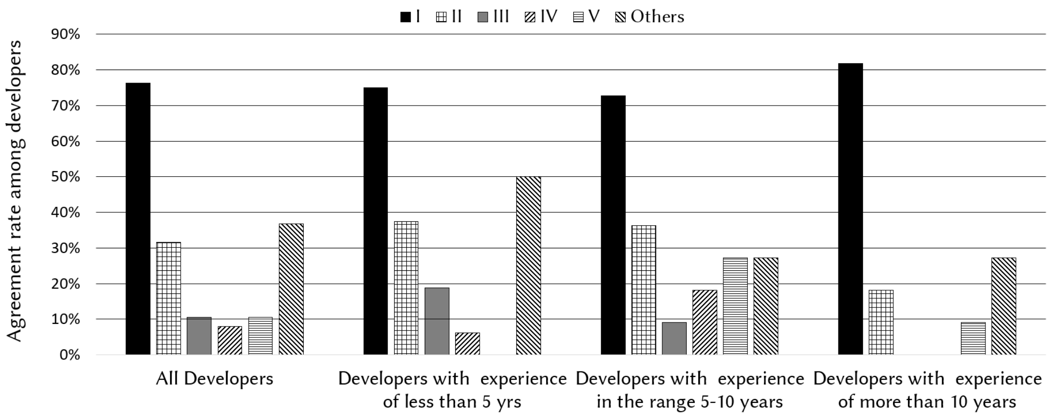


Figure 6.2: Infograph displaying what types of Twitter accounts developers across different experience levels prefer to follow. For descriptions of categories I-V, please refer to Table 6.1.

After performing the open card sort on the responses provided by the 36 respondents, we were able to identify 5 prominent categories apart from *Others*. These categories are shown in Table 6.1. Figure 6.2 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 work, 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.

6.3 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.

6.3.1 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/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 [68]. 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 which 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 following three categories:

- *Conversational tweets (CT)* are tweets that mention at least one Twitter user.
- *Retweeted tweets (RT)* 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/her followers.
- *Original Tweet (OT)* are the non RT and CT tweets that are produced by a Twitter user.

Based on the above concepts, Table 6.2 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 [68]. These features are based on the feature components introduced in Table 6.2. All of them are calculated for each user with respect to a particular domain. We further sub categorize the

Table 6.2: 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/her domain-related tweets, where domain is *d*
M2(u,d)	Number of unique domain-related users mentioned by a user *u* in his/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*

content features into categories of *Content Strength* and *Content Popularity*. We describe the sub categories 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/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].

- *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:

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

- *Non-Chat Signal*: Non-Chat Signal (NCS) captures how much a user posts on the domain and how much he/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 [68] 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 [68], 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/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[2]. Then each tweet *i* is represented as a vector of words s_i which contains the remaining non stop 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:

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

  In this equation, n is the total number 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/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, higher LR score might improve the likelihood of a topic-related tweet being useful to other developers. Twitter has a limit of 140 characters

[2]`http://www.ranks.nl/stopwords`

per tweet and each URL shared consumes 23 characters, so a tweet can at the maximum contain 5 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. 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 space or punctuation. So any hashtag will atleast 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. 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 some of their devoted followers always retweet the content. To dampen the impact of such users the multiplication by 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 regards to the domain of interest. 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 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, 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 first component.

  The above steps ensure that the Mention Impact(MI) we calculate for a user is based on his/her merit and not as a result of him/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 [68]. This feature can take values in the interval $[0,\infty)$.

- *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 $[0,\infty)$.

6.3.2 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 that benefit from his/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 [132, 12, 67, 27]. We use some of the centrality indicators proposed in [132, 12], 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 [67]. 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 of *Centrality Scores* and *Absolute Scores*. We describe the sub categories 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 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 [89]. 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 an 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 ratio of number of other users it follows to the total number of users in the network. Intuitively experts on Twitter have 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 [67]. The PR algorithm computes a probability to represent the likelihood that 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 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 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 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 which 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.

6.3.3 Profile Features

A Twitter user can specify his/her biodata and include a reference to his/her webpage in his/her Twitter account. This information can help us to better characterize a Twitter user. Keywords such as $developer, architect, consultant$, etc. 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 $recruiter, headhunter$, etc. help to identify and eliminate accounts related to hiring firms. These accounts are not preferred by most developers as discussed in Section 6.2.

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/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*[3]. 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 sub-steps:

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", etc, 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 [75]

[3]`http://www.crummy.com/software/BeautifulSoup/`

In the end, for each user, we construct two feature vectors; one to represent his/her biodata and the other to represent his/her webpage. Each feature corresponds to a pre-processed 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 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 6.4.2. 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 etc. and whose authenticity has been confirmed [4]. 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 measures indicates from 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 more close 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] `https://support.twitter.com/articles/119135`

6.3.4 GitHub Features

Some Twitter users include links to their GitHub profiles in their webpages. GitHub is one of the popular code and repository holding website having over 21.1 million repositories held by over 9 million users [28]. 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 5 basic GitHub features.

- *IsGhMentioned*: This feature indicates whether a Twitter user includes a link to his/her GitHub profile in his/her webpage. Intuitively a software expert will want to publish a link to his/her GitHub profile on his/her webpage to highlight his/her work and possibly to find interested people to join the projects he/she is championing on GitHub. A newbie or a non-expert developer is likely not to have a GitHub profile and even if he/she has one he/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 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 the number of 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 a GitHub repositories which are generally entire projects in themselves. A user who has 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" else the feature is assigned a value of 0.

6.4 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.

6.4.1 Approach Overview

Figure 6.3 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 out 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, 85, 104], wherein initially we create a seed list of popular Twitter users who are

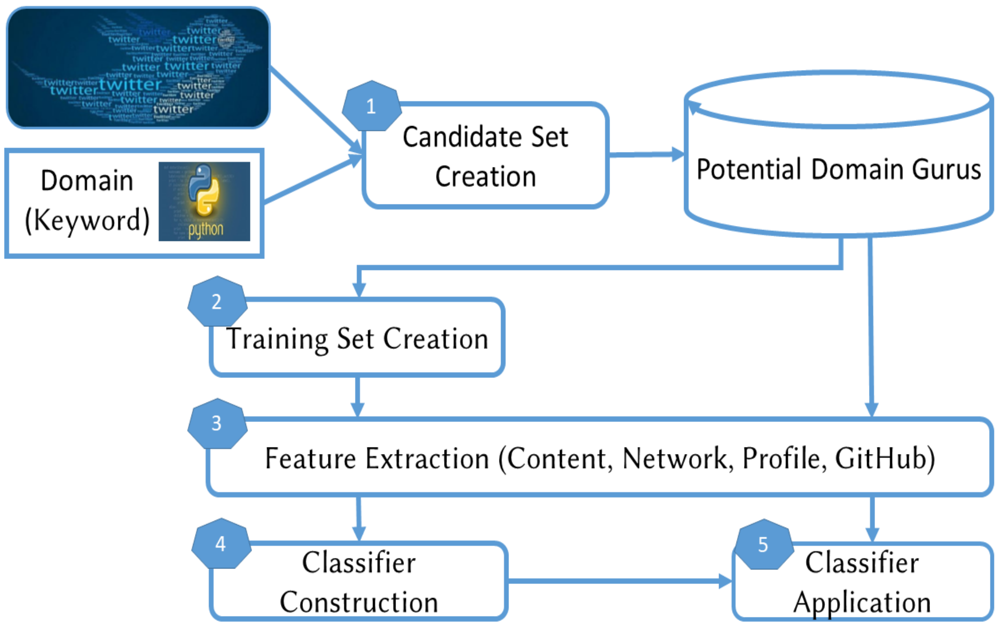


Figure 6.3: Framework of Our Recommendation System

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 seed list. Next as we need to find users who are related to a domain, we filter Twitter users who post less 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 6.5.1.

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 6.5.1), 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 6.3 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.

6.4.2 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/her biodata (i.e., $P(Guru|Biodata)$), the probability of a Twitter user to be not a specialized guru given his/her biodata (i.e., $P(\neg Guru|Biodata)$), the probability of a Twitter user to be a specialized guru given his/her webpage (i.e., $P(Guru|Webpage)$), and the probability of a Twitter user to be not a specialized guru given his/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., [135].

Constructing a Unified Discriminative Model:

After we have processed the profile features, we combine the 4 probabilities with the 10 content features, 4 other profile features, 9 network features and the 5

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 Naive Bayes Multinomial (NBM) on the 32 features from the four families (i.e., content, network, profile and GitHub).

6.5 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.

6.5.1 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[5]; this list of seed users was used by previous studies [1, 85, 104]. 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[6][7][8][9]. From these blogs, we are able to extract 48 unique users. These 48 users were then merged with the previous 100 users that results in a final set of 139 users (after removing duplicates) which we refer to as $uSeed$.

[5]`http://www.noop.nl/2009/02/twitter-top-100-for-softwaredevelopers.html`

[6]`https://www.untapt.com/blog/2015/11/25/developers-to-follow-on-twitter/`

[7]`https://www.thebalance.com/programmers-on-twitter-2072010`

[8]`http://zartis.com/ten-software-developers-follow-twitter/`

[9]`http://www.techworld.com/picture-gallery/social-media/people-all-developers-should-follow-on-twitter-3644265/`

Table 6.3: 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

We than 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 contents 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 one-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 work 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[10].

We evaluated the effectiveness of our approach by recommending software gurus for four domains: JavaScript, Android, Python and Linux. Javascript and Python

[10] `https://techcrunch.com/2016/08/15/linkedin-sues-scrapers/`

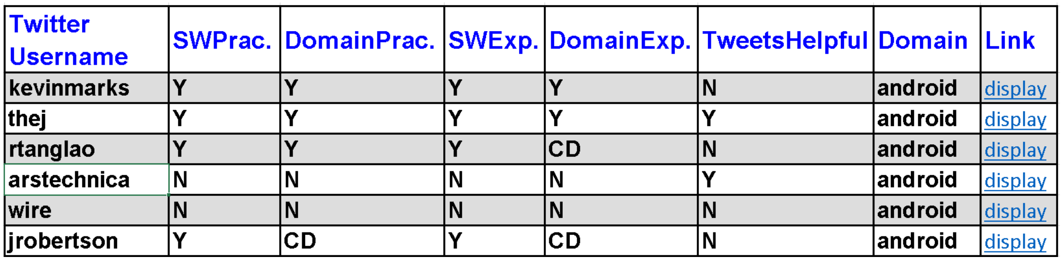
Table 6.4: Inter Rater Agreement

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

are programming languages while Android and Linux are operating systems. We chose these domains since among tweets in our dataset, these domains were well represented. Indeed, in our dataset, JavaScript-related tweets are more than any other domain related tweets. Another consideration was that we were easily able to find people to label the data as gurus and non-gurus 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 less 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 6.3. 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 6.3 summarizes basic statistics of our dataset.

Next, we asked 6 PhD students majoring in Computer Science and 2 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 five 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 2-3 persons independently. A participant was assigned to a domain only if he/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 inter-rater agreement. For cases that they disagree, the labelers sat down together to discuss and decide final labels.

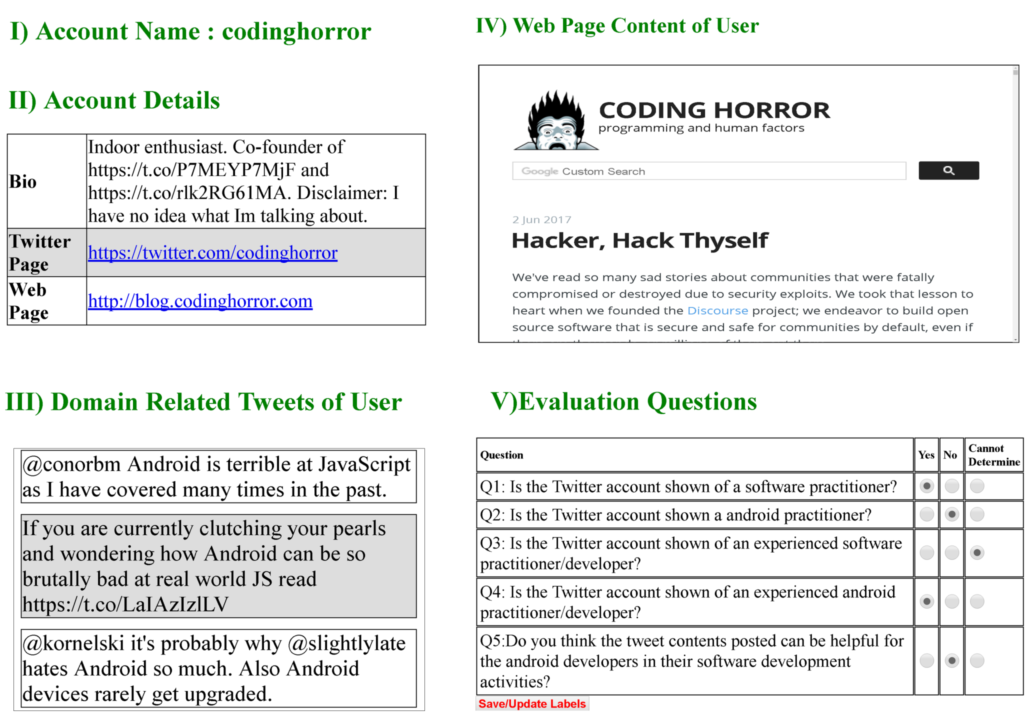
To better support the labeling process, we provided a web-based labelling system for the participants. Figure 6.4 shows the main page of our labelling system, which contains a list of Twitter users who need to be labeled. For each user, the participant had to click the "display" button to enter to an evaluation page. Figure 6.5 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 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; V) evaluation questions.



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

Figure 6.4: Main Page of Our Labelling System

We asked participants to answer five questions in part V based on the information shown in parts I-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/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


I) Account Name : codinghorror

II) Account Details

Bio	Indoor enthusiast. Co-founder of https://t.co/P7MEYP7MjF and https://t.co/rlk2RG61MA. Disclaimer: I have no idea what Im talking about.
Twitter Page	https://twitter.com/codinghorror
Web Page	http://blog.codinghorror.com

III) Domain Related Tweets of User

@conorbm Android is terrible at JavaScript as I have covered many times in the past.

If you are currently clutching your pearls and wondering how Android can be so brutally bad at real world JS read https://t.co/LaIAzIzlLV

@kornelski it's probably why @slightlylate hates Android so much. Also Android devices rarely get upgraded.

IV) Web Page Content of User

CODING HORROR
programming and human factors

Google Custom Search

2 Jun 2017

Hacker, Hack Thyself

We've read so many sad stories about communities that were fatally compromised or destroyed due to security exploits. We took that lesson to heart when we founded the Discourse project; we endeavor to build open source software that is secure and safe for communities by default, even if

V)Evaluation Questions

Question	Yes	No	Cannot Determine
Q1: Is the Twitter account shown of a software practitioner?	●	○	○
Q2: Is the Twitter account shown a android practitioner?	○	●	○
Q3: Is the Twitter account shown of an experienced software practitioner/developer?	○	○	●
Q4: Is the Twitter account shown of an experienced android practitioner/developer?	●	○	○
Q5:Do you think the tweet contents posted can be helpful for the android developers in their software development activities?	○	●	○

Save/Update Labels

Figure 6.5: Evaluation Page for a Twitter User

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 answer for question 4 as “Yes” and answer for question 5 as “No”, we labeled them as “Others”. For users where answer to question 4 is “No”, we labeled them also as “Others”. We omitted the rest of users from final dataset, since their labels cannot be reliably determined.

The inter-rater agreement scores for answers to question 4 and 5 over all domains are shown in Table 6.4. We used Cohen’s Kappa [20] to measure inter-rater reliability for the labeling task. A Cohen’s Kappa score less or equal to zero is considered as no agreement, between 0.01-0.20 is considered as none to slight agreement, between 0.21 and 0.40 is considered as fair agreement, between 0.41 and

0.60 is considered as moderate agreement, between 0.61 and 0.80 is considered as substantial agreement, and between 0.81 and 1.0 is considered as almost perfect agreement [125, 60]. We can see from Table 6.4 that except for the Linux dataset the agreement is at least moderate. For the Linux dataset, the agreement is still fair.

Table 6.5: Number of Specialized Software Gurus

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

Table 6.5 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 one month period, and whose labels can be reliably determined. For example, for "Python" domain, initially a total of 3,710 Twitter users had posted at least 1 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 "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[11] so the number of "JavaScript" gurus on Twitter are 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.

[11] `https://insights.stackoverflow.com/survey/2017#technology`

6.5.2 Experiment Setting

Implementation-Details: We use the implementation of Multinomial Naive Bayes[12] provided as part of sklearn [13, 70].

Evaluation Metrics: We use three standard metrics, namely precision, recall, and F-Measure, which have been used in many past studies, e.g., [106, 135]. 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/she is correctly predicted as such (true positive, TP); the user is not a specialized software guru, however he/she is wrongly predicted as a specialized software guru (false positive, FP); the user is a specialized software guru, however he/she is not predicted as such (false negative, FN); or the user is not a specialized software guru, and he/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 1 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

[12]`http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html`

- Our first baseline is the approach proposed by Pal and Counts [68] 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[13] [70] 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 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 we run Pal and Counts 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[14]. Klout is a system which calculates influence score of social accounts across multiple social networks [77]. 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[15] 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 [91, 77] 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 on the domain. We refer to this baseline as KL.

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

6.5.3 Research Questions and Results

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

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 6.5.2.

Results: Table 6.6 shows the precision, recall, and F-measure of our approach on four different domains. From Table 6.6, 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 6.6 we can see that the F-Measure for Linux domain is low, achieving a value of 0.522. To identify the reasons for 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, etc., 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 which show results with F-Measure in the range of 0.5-0.7 [19, 83, 144, 115, 119]. 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' work [68] by proposing new features (i.e., 9 network, 8 profile features and 5 GitHub features)

Table 6.6: 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

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 6.5.2 to compute the precision, recall, and F-Measure of Pal and Counts' approach [68] and Klout approach [77]. We then compare and contrast their evaluation scores with those of ours.

Table 6.7: 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%

Results: Table 6.7 shows the performance of the two variants of Pal and Counts' approach and the Klout baseline on the four different domains. From Table 6.7, we observe that our approach (shown in Table 6.6) can consistently achieve 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 7.63-166.10%. The Klout baseline is also outperformed by our approach on all domains by 18.79%-31.47%.

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

Motivation and Approach: In our approach we use 32 different features to characterize a Twitter user, i.e., 10 content features, 9 network features, 8 profile features, and 5 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 6.8 shows the various feature combinations that we have evaluated. The F-Measure scores shown in 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 [68]. Next to measure the strength of each category of features, we remove one category at a time and then calculate the corresponding F-Measures. *ALL-GitHub* row refers to the setting where we use all features except those belonging to 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 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 6.8 are

Table 6.8: 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%

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 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 bio of users, they contain words which can be used to identify experts. Also in *Profile* category there are features which 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, 9 network features, 8 profile features, and 5 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 6.6 and the percentage drop was computed. The features which on removal cause the highest percentage drop in F-Measure are considered as the most important. These top-10 features for each domain are shown in Table 6.9.

Results: In Table 6.9, 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 they appear in the top-10 lists of the four domains. Table 6.10 shows the features that have appeared in the top-10 lists of at least two domains.

From Table 6.10, 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 3 domains. However, only the Profile feature PosBio is present in top-5 ranks across the 3 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

Table 6.9: 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 6.10: 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

the list for at-least 2 domains.

From Table 6.9 it can be observed the Profile features are the most frequent among Top-10 features and at relatively higher ranks. This is in line with the results observed in Table 6.8 where removing the Profile category had caused the highest drop in F-Measure. The probabilities extracted from user's 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?*

Table 6.11: 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

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.

Results: Table 6.11 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 *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 6.6 (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 *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 *cross-domain* result is significantly different from F-Measure obtained in *within-domain* setting we performed the Mann-Whitney U test [56] on the means of F-Measures obtained in *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 *cross-domai*n 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 Linux system).

6.5.4 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 inter-rater agreement for the labeling task using the measure of Cohen’s Kappa [20]. As can be seen from Table 6.4, 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, etc. 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 work, 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 6.8). 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[16]. 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) users have BitBucket links in their profiles. Note that the collection of users in our dataset is not biased in any way towards GitHub[17]. Many past studies have also focused on

[16]`https://octoverse.github.com/`

[17]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 6.4

GitHub due to its popularity [122, 120, 96, 93, 89, 84].

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 6.5.1, 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 small change in the F-Measure (an increase in F-Measure by 0.5-4.3% 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 0.99. We found little change in performance (a decrease in F-Measure by 0-1.21%) when we vary the threshold from 0.75 to 0.99 (using a step of 0.03).

The generalizability of our results may also be impacted in case we wanted to give recommendations to users who are from a particular geography, or whose primary language of communication is other than English. The language of our survey request and response, as well as final user study, was in English, and the persons who labeled the data for experiments were also English-speaking users. This may limit our approach to be usable only for English users. However, the three feature categories namely Content, Profile, and Network should still be helpful in finding experts when the constraints of language or geography are applied. On the other hand, the feature category of GitHub may need to be expanded to include websites which may be more popular in a given geography, e.g. `https://coding.net/` is very popular in some geographies such as China. We plan to address this threat in future work, by accounting for the user language as well as the geography of the user while giving the final recommendation.

Threats to construct validity relates to the suitability of our evaluation metric. In

Table 6.12: Feature Strength vs 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

this work, we use precision, recall, and F-Measure. These metrics are well-known and have been used in many past studies, e.g., [49, 135, 145]. 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 6.6.2. The study finds the recommendations provided by our approach are at least 25.93% more accurate than the baselines.

6.6 Discussion

6.6.1 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 baseline approach on all features combined.

Table 6.12 shows the results of our experiments. From Table 6.12 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 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 [68] approach using features from all categories. Table 6.12 shows that the F-Measure drops down to 0.497 for PC^{Ev} baseline variant and 0.403 for PC^{Tr+Ev} variant. This shows that our two-stage classification approach is able to achieve better performance over the baseline approach of [68] when all the features are used.

6.6.2 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 "Android" domain that we have collected earlier – see Table 6.5. We divide this dataset into 2 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 the 4 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; the question being: "Are you interested in following the above Twitter account, so that following it may help you in getting 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 author of the dissertation 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.

Table 6.13: 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 6.14: User Study Results

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

The answers provided by each user were converted into binary ratings following the conversion table shown in Table 6.13. To evaluate the results of our user study, we make use of Normalized Discounted Cumulative Gain (NDCG) [39]. NDCG is commonly used to measure the performance of information retrieval and recommendation systems [51]. 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 6.14 shows the results of our user study for each of the four approaches (ours and the 3 baselines). We can see from Table 6.14 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.

6.6.3 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 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 [89, 96, 143]. While this method may work well

and can result in response rates greater than 15%, but 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 [50] 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 [30]. 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.

6.7 Conclusion and Future Work

Twitter is becoming increasingly popular these years 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 [89]. 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 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 which 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 0.522-0.820 on the four domains. Our approach can improve the F-measures achieved by baseline approaches [77, 68] 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. Also we plan to conduct further user studies to understand the difference in perspectives of developers based on their language and/or geography and incorporate that into the current approach to make it more robust.

Chapter 7

Conclusion and Future Work

7.1 Summary of Contribution

Software developers use various social media channels and tools to support their software development activities. However, developers face some challenges when they use these channels. Some examples of these challenges are information overload, maintaining relevant connections, continuous distraction, etc. This dissertation tries to address some of these challenges with respect to Twitter, a popular social media channel used by developers [89, 96]. The first three works deal with understanding the popular software engineering content on Twitter and helping developers discover such content. The last work is focused on how to find software experts who produce such content. A summary of completed works is described below.

- **Understanding Popular Software Engineering Content Produced in Social Networks**: This work is an exploratory study in which the trending topics in Twitter related to software engineering [86] were explored. It was found that article and multimedia sharing, technical discussion, and new version releases are the top-3 most popular categories related to software engineering on Twitter. The categories developed in this work add to the theory of knowledge in programming and can be put into the category of externalized knowledge [64]. The findings of this study can help guide the research in systems and techniques which focus on mining information and knowledge related to software engineering from Twitter.

- **Automatic Identification of Software Relevant Content in Social Media**: This work proposed a novel approach named NIRMAL, which can automatically identify software relevant tweets from a collection or stream of tweets [85]. The approach was able to achieve accuracy@K scores of up to 0.900, and performed better than keyword based approach by up to 31%. This approach can help developers to easily discover interesting and relevant software related information from microblogs that a developer gets exposed to. This work tries to address the challenge of information overload which is considered a major barrier to adoption of Twitter among software developers [89].

- **Mining Informative Online Resources Shared by Developers on Social Media**: In this work, Twitter was leveraged to find informative and relevant resources related to a particular domain of interest [87]. The work proposed 14 features to characterize each URL by considering contents of webpage pointed by it, contents and popularity of tweets mentioning it, and the popularity of users who shared the URL on Twitter. Also, evaluation of an unsupervised and a supervised approach was performed to find and rank URLs harvested from Twitter. The results of our experiments on tweets generated by a set of 85,171 users over a one-month period highlight that the proposed unsupervised and supervised approaches can achieve a high Normalized Discounted Cumulative Gain (NDCG) score of 0.719 and 0.832 respectively.

- **Recommending Experts in the Software Engineering Twitter Space**: In this work an approach has been proposed to identify software experts on Twitter [88]. First, an open-ended online survey was conducted with developers who use Twitter to support their software development activities. A qualitative analysis of the survey responses showed that developers are interested in following specialized software experts, who also generate technical content. Based on this insight, an approach based on 32 features was designed to classify a given Twitter user as a *Software Expert* or *Others*. For evaluation, a binary labeled dataset of 614 users with labels being *Software Expert* and *Others* was created, and the proposed approach achieves F-Measure scores of 0.522-0.820 (for four domains i.e., JavaScript, Python, Android, and Linux) on the task of finding software experts among users in this dataset. Also when compared to baseline approaches the proposed method is able to achieve an improvement of at least 7.63% over the baselines. The proposed approach can help developers to address the challenge of finding specialized accounts on Twitter highlighted in [89]. Also, the approach can be used to strengthen information mining techniques such as those proposed in [87, 85] to recommend more relevant content.

7.2 Future Directions

Despite the benefit brought by Twitter, its enormous size poses a number of challenges for its users, including software developers. Singer et al. highlighted *information overload* and finding *relevant accounts to follow* as the two main challenges faced by developers when using Twitter to support their software development activities [89]. The works done as part of this dissertation try to address these challenges. The techniques described in Chapters 4 and 5 of this thesis show promise in extracting software related content from Twitter. In its current form, the content extracted is general in nature so there is scope of future research on developing techniques to classify the extracted content into fine grained categories which can be used to support various software evolution processes. The Chapter 6 of this thesis focuses on helping developers to identify experts they can follow on Twitter. This work can be complemented by a study of what kind of accounts software developers like to unfollow on Twitter. Based on this, an approach may be developed to recommend accounts to unfollow on Twitter, helping them to maintain a more software engineering relevant network on Twitter. Some of these directions are briefly discussed below.

7.2.1 Automated Cataloging of Software Engineering Tweets

The works performed as a part of current thesis mainly focus on mining software relevant information from Twitter. However, the information extracted can be related to many categories. The tweets may be related to categories such as articles, technical discussions, promotions, opinions etc. [76, 86]. As many tweets relate to discussion and opinions, one future research direction is to develop techniques which summarize such discussions and/or opinions about various software artifacts such as software libraries, API's, packages etc. Tweets related to API's may also be used to support automated API documentation techniques such as envisaged in [110, 99, 81]. Previous studies have also found that many software related tweets relate to various stages of software evolution, such as software requirements [32, 133], bug fixing [25] etc. Thus, an interesting direction to work in future is to develop a general classifier which can categorize the software engineering information extracted from Twitter into various such categories. One other dimension that can be looked into future is the use of latest techniques such as Word2Vec [61, 62], to improve the accuracy of techniques such as NIRMAL which was proposed in Chapter 4. A recent work in this direction has been proposed in [100]. Overall, such techniques can contribute to developing an automatic cataloging system which categorizes software engineering tweets into various categories.

7.2.2 Understanding the Unfollow Behavior of Developers in Software Engineering Twitter Space

Singer et al. had found that one of the ways software developers maintain a relevant network on Twitter is to regularly unfollow users [89]. Bases on this intuition, further research can be conducted to understand what kind of accounts do software developers unfollow. One of the ways to do this is to conduct a survey similar to those performed in [89, 88]. In this survey developers on Twitter can be asked about their unfollow preferences. One other way is doing an observational study of unfollow behaviour of some identified software developers on Twitter, as has been done in [41]. Insights gathered from such studies can be used either to develop an approach which can recommend accounts to developers which they may unfollow on Twitter, or to use the insights to strengthen the expert recommendation technique that has been proposed in Chapter 6.

.

List of Publications

Main Publications

The work performed as a part of this dissertation has been presented in the following publications.

Abhishek Sharma, Yuan Tian, Agus Sulistya, Dinusha Wijedasa, and David Lo. "Recommending Who to Follow in the Software Engineering Twitter Space." In *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(4), (2018).

Abhishek Sharma, Yuan Tian, Agus Sulistya, David Lo, and Aiko Fallas Yamashita. "Harnessing Twitter to Support Serendipitous Learning of Developers." In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017.

Abhishek Sharma, Yuan Tian, and David Lo. "What's Hot in Software Engineering Twitter Space?" In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015.

Abhishek Sharma, Yuan Tian, and David Lo. "NIRMAL: Automatic Identification of Software Relevant Tweets Leveraging Language Model." In *IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2015 .

Others

The following are the publications in which the author has contributed, but which are not related to this dissertation.

Yuan Tian, Ferdian Thung, **Abhishek Sharma**, and David Lo. "APIBot: Question Answering Bot for API Documentation." In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017.

Xia, Xin, David Lo, Lingfeng Bao, **Abhishek Sharma**, and Shanping Li. "Personality and Project Success: Insights from a Large-Scale Study with Professionals." In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017

Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya and David Lo. "Cataloging GitHub Repositories." In *21st International Conference on Evaluation and Assessment in Software Engineering Conference (EASE)*, 2017.

Agus Sulistya, **Abhishek Sharma**, and David Lo.“Spiteful, One-Off, and Kind: Predicting Customer Feedback Behavior on Twitter.” In *International Conference on Social Informatics (SocInfo)*, 2016.

Doyen Sahoo, **Abhishek Sharma**, Steven C.H. Hoi, and Peilin Zhao. “Temporal Kernel Descriptors for Learning with Time-sensitive Patterns.” In *2016 SIAM International Conference on Data Mining (SDM)*, 2016.

Bibliography

[1] P. Achananuparp, I. N. Lubis, Y. Tian, D. Lo, and E.-P. Lim. Observatory of trends in software related microblogs. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 334–337, 2012.

[2] M. Allamanis and C. Sutton. Mining source code repositories at massive scale using language modeling. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 207–216, 2013.

[3] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, and M. A. Gerosa. How modern news aggregators help development communities shape and share knowledge. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, pages 499–510, 2018.

[4] J. A. Aslam and M. Montague. Models for metasearch. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 276–284, 2001.

[5] A. S. Badashian, A. Esteki, A. Gholipour, A. Hindle, and E. Stroulia. Involvement, contribution and influence in github and stack overflow. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering (CASCON)*, pages 19–33, 2014.

[6] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering (EMSE)*, pages 619–654, 2014.

[7] B. Bazelli, A. Hindle, and E. Stroulia. On the personality traits of stackoverflow users. In *29th IEEE International Conference on Software Maintenance (ICSM)*, pages 460–463, 2013.

[8] A. Begel, R. DeLine, and T. Zimmermann. Social media for software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 33–38, 2010.

[9] S. Beyer and M. Pinzger. Grouping android tag synonyms on stack overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR)*, pages 430–440, 2016.

[10] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft. Building reputation in stackoverflow: an empirical investigation. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 89–92, 2013.

[11] G. Bougie, J. Starke, M.-A. Storey, and D. M. German. 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 (Web2SE)*, pages 31–36, 2011.

[12] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, pages 136–145, 2008.

[13] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013.

[14] B. A. Campbell and C. Treude. Nlp2code: Code snippet content assist via natural language tasks. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 628–632, 2017.

[15] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological Methods & Research*, pages 294–320, 2013.

[16] C. Chen, S. Gao, and Z. Xing. Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 338–348, 2016.

[17] C. Chen and Z. Xing. Mining technology landscape from stack overflow. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, page 14, 2016.

[18] C. Chen and Z. Xing. Similartech: automatically recommend analogical libraries across different programming languages. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 834–839, 2016.

[19] M. Choetkiertikul, D. Avery, H. K. Dam, T. Tran, and A. Ghose. Who will answer my question on stack overflow? In *24th Australasian Software Engineering Conference (ASWEC)*, pages 155–164, 2015.

[20] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, pages 37–46, 1960.

[21] J. Corbin, A. Strauss, and A. L. Strauss. *Basics of qualitative research*. 2014.

[22] D. Correa and A. Sureka. Chaff from the wheat: characterization and modeling of deleted questions on stack overflow. In *Proceedings of the 23rd international conference on World wide web (WWW)*, pages 631–642, 2014.

[23] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (CSCW)*, pages 1277–1286, 2012.

[24] Q. Diao and J. Jiang. A unified model for topics, events and users on twitter. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1869–1879, 2013.

[25] M. El Mezouar, F. Zhang, and Y. Zou. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. *Empirical Software Engineering (EMSE)*, pages 1704–1742, 2018.

[26] J. Escobar-Avila, E. Parra, and S. Haiduc. Text retrieval-based tagging of software engineering video tutorials. In *IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 341–343, 2017.

[27] L. C. Freeman. Centrality in social networks conceptual clarification. *Social networks*, pages 215–239, 1978.

[28] GitHub. About github inc. 2015. [Online; accessed 29-Aug-2015].

[29] S. Gottipati, D. Lo, and J. Jiang. Finding relevant answers in software forums. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 323–332, 2011.

[30] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 233–236, 2013.

[31] S. Grant and B. Betts. Encouraging user behaviour with achievements: an empirical study. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 65–68, 2013.

[32] E. Guzman, R. Alkadhi, and N. Seyff. A needle in a haystack: What do twitter users say about software? In *IEEE 24th International Requirements Engineering Conference (RE)*, pages 96–105, 2016.

[33] E. Guzman, R. Alkadhi, and N. Seyff. An exploratory study of twitter messages about software applications. *Requirements Engineering*, 22(3):387–412, 2017.

[34] E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20, 2017.

[35] A. Halavais, K. H. Kwon, S. Havener, and J. Striker. Badges of friendship: Social influence and badge acquisition on stack overflow. In *47th Hawaii International Conference on System Sciences (HICSS)*, pages 1607–1615, 2014.

[36] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *34th International Conference on Software Engineering (ICSE)*, pages 837–847, 2012.

[37] W. Hudson. Card sorting. *The Encyclopedia of Human-Computer Interaction, 2nd Ed.*, pages 92–101, 2013.

[38] Y. Inagaki, J. Bian, and Y. Chang. An effective general framework for localized content optimization. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 65–66, 2013.

[39] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, pages 422–446, 2002.

[40] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang. Why and how developers fork what from whom in github. *Empirical Software Engineering (EMSE)*, pages 547–578, 2017.

[41] J. Jiang, D. Lo, Y. Yang, J. Li, and L. Zhang. A first look at unfollowing behavior on github. *Information and Software Technology (IST)*, 2018.

[42] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories (MSR)*, pages 92–101, 2014.

[43] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401, 1987.

[44] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 96–107, 2016.

[45] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering (TSE)*, pages 1–1, 2017.

[46] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, pages 604–632, 1999.

[47] R. Kuhn and R. De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

[48] H. Kwak, C. Lee, H. Park, and S. B. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 591–600, 2010.

[49] A. Lazar, S. Ritchey, and B. Sharif. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pages 308–311, 2014.

[50] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik. 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 (CSCW)*, pages 333–336, 2016.

[51] T. Liu. *Learning to Rank for Information Retrieval*. 2011.

[52] D. Lo, N. Nagappan, and T. Zimmermann. How practitioners perceive the relevance of software engineering research. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering(FSE)*, pages 415–425, 2015.

[53] Y. Ma, S. Fakhoury, M. Christensen, V. Arnaoudova, W. Zogaan, and M. Mirakhorli. Automatic classification of so ware artifacts in open-source applications. 2018.

[54] L. MacLeod, A. Bergen, and M.-A. Storey. Documenting and sharing software knowledge using screencasts. *Empirical Software Engineering (EMSE)*, pages 1478–1507, 2017.

[55] L. MacLeod, M.-A. Storey, and A. Bergen. 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 (ICPC)*, pages 104–114, 2015.

[56] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.

[57] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[58] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[59] J. B. Marino, R. E. Banchs, J. M. Crego, A. de Gispert, P. Lambert, J. A. Fonollosa, and M. R. Costa-Jussà. N-gram-based machine translation. *Computational Linguistics*, pages 527–549, 2006.

[60] M. L. McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, pages 276–282, 2012.

[61] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[62] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[63] A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu. Among the machines: Human-bot interaction on social q&a websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1272–1279, 2016.

[64] P. Naur. Programming as theory building. *Microprocessing and microprogramming*, 15(5):253–261, 1985.

[65] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. A statistical semantic language model for source code. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 532–542, 2013.

[66] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli. Are bullies more productive? empirical study of affectiveness vs. issue fixing time. In *IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*, pages 303–313, 2015.

[67] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[68] A. Pal and S. Counts. Identifying topical authorities in microblogs. In *Proceedings of the fourth ACM international conference on Web search and data mining (WSDM)*, pages 45–54, 2011.

[69] E. Parra, J. Escobar-Avila, and S. Haiduc. Automatic tag recommendation for software development video tutorials. In *Proceedings of the 26th Conference on Program Comprehension (ICPC)*, pages 222–232, May 27-28 2018.

[70] 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. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research (JMLR)*, 12:2825–2830, 2011.

[71] E. Poché, N. Jha, G. Williams, J. Staten, M. Vesper, and A. Mahmoud. Analyzing user comments on youtube coding tutorial videos. In *Proceedings of the 25th International Conference on Program Comprehension (ICPC)*, pages 196–206, 2017.

[72] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza. Too long; didn't watch!: extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 261–272, 2016.

[73] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, B. Russo, S. Haiduc, and M. Lanza. Codetube: Extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICPC)*, pages 645–648, 2016.

[74] L. Ponzanelli, G. Bavota, A. Mocci, R. Oliveto, M. Di Penta, S. C. Haiduc, B. Russo, and M. Lanza. Automatic identification and classification of software development video tutorial fragments. *IEEE Transactions on Software Engineering (TSE)*, pages 1–1, 2017.

[75] M. F. Porter. An algorithm for suffix stripping. *Program*, pages 130–137, 1980.

[76] P. K. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E.-P. Lim. Automatic classification of software related microblogs. In *28th IEEE International Conference on Software Maintenance (ICSM)*, 2012.

[77] A. Rao, N. Spasojevic, Z. Li, and T. DSouza. Klout score: Measuring influence across multiple social networks. In *IEEE International Conference on Big Data (Big Data)*, pages 2282–2289, 2015.

[78] A. Rastogi and N. Nagappan. Forking and the sustainability of the developer community participation–an empirical investigation on outcomes and reasons. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 102–111, 2016.

[79] A. Rastogi and N. Nagappan. On the personality traits of github contributors. In *IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 77–86, 2016.

[80] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, editors. *Recommendation Systems in Software Engineering*. Springer, 2014.

[81] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, et al. On-demand developer documentation. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 479–483, 2017.

[82] J. Saldaña. *The coding manual for qualitative researchers*. Sage, 2015.

[83] G. Santos, K. V. Paixão, N. Anquetil, A. Etien, M. de Almeida Maia, and S. Ducasse. Recommending source code locations for system specific transformations. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 160–170, 2017.

[84] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo. Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 314–319, 2017.

[85] A. Sharma, Y. Tian, and D. Lo. Nirmal: Automatic identification of software relevant tweets leveraging language model. In *IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 449–458, 2015.

[86] A. Sharma, Y. Tian, and D. Lo. What's hot in software engineering twitter space? In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 541–545, 2015.

[87] A. Sharma, Y. Tian, A. Sulistya, D. Lo, and A. F. Yamashita. Harnessing twitter to support serendipitous learning of developers. In *IEEE 24th International Conference one Analysis, Evolution and Reengineering (SANER)*, pages 449–458, 2017.

[88] A. Sharma, Y. Tian, A. Sulistya, D. Wijedasa, and D. Lo. Recommending who to follow in the software engineering twitter space. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(4):16, 2018.

[89] L. Singer, F. M. F. Filho, and M. D. Storey. Software engineering at the speed of light: how developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 211–221, 2014.

[90] E. K. Smith, C. Bird, and T. Zimmermann. Build it yourself!: homegrown tools in a large software company. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, pages 369–379, 2015.

[91] N. Spasojevic, J. Yan, A. Rao, and P. Bhattacharyya. Lasta: Large scale topic assignment on multiple social networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 1809–1818, 2014.

[92] A. Stolcke et al. Srilm-an extensible language modeling toolkit. In *INTERSPEECH*, 2002.

[93] M.-A. Storey. Msr 2012 keynote: The evolution of the social programmer. In *9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 140–140, 2012.

[94] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r) evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*, pages 100–116, 2014.

[95] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 359–364, 2010.

[96] M.-A. Storey, A. Zagalsky, F. Figueira Filho, L. Singer, and D. M. German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering (TSE)*, pages 1–1, 2017.

[97] A. Strauss and J. Corbin. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE Publications, 1990.

[98] A. Strauss and J. M. Corbin. *Grounded theory in practice*. Sage, 1997.

[99] S. Subramanian, L. Inozemtseva, and R. Holmes. Live api documentation. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 643–652, 2014.

[100] A. Sulistya, G. A. A. Prana, A. Sharma, D. Lo, and C. Treude. Sieve: Helping developers sift wheat from chaff via cross-platform analysis. *arXiv preprint arXiv:1810.13144*, 2018.

[101] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang. Network structure of social coding in github. In *17th european conference on Software maintenance and reengineering (CSMR)*, pages 323–326, 2013.

[102] F. Thung, D. Lo, and J. Lawall. Automated library recommendation. In *20th Working Conference on Reverse Engineering (WCRE)*, pages 182–191, 2013.

[103] Y. Tian, P. Achananuparp, I. N. Lubis, D. Lo, and E.-P. Lim. What does software engineering community microblog about? In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 247–250, 2012.

[104] Y. Tian and D. Lo. An exploratory study on software microblogger behaviors. In *2014 IEEE 4th Workshop on Mining Unstructured Data (MUD)*, pages 1–5, 2014.

[105] Y. Tian, D. Lo, and J. Lawall. Automated construction of a software-specific word similarity database. In *Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 44–53, 2014.

[106] Y. Tian, D. Lo, X. Xia, and C. Sun. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering (EMSE)*, pages 1354–1383, 2014.

[107] Y. Tian, F. Thung, A. Sharma, and D. Lo. Apibot: Question answering bot for api documentation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 153–158, 2017.

[108] C. Treude, O. Barzilay, and M. D. Storey. How do programmers ask and answer questions on the web? In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 804–807, 2011.

[109] C. Treude, F. Figueira Filho, B. Cleary, and M.-A. Storey. Programming in a socially networked world: the evolution of the social programmer. *The Future of Collaborative Software Development*, pages 1–3, 2012.

[110] C. Treude and M. P. Robillard. Augmenting api documentation with insights from stack overflow. In *IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 392–403, 2016.

[111] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th international conference on Software engineering (ICSE)*, pages 356–366, 2014.

[112] J. Tsay, L. Dabbish, and J. Herbsleb. Let's talk about it: evaluating contributions through discussion in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering (FSE)*, pages 144–154, 2014.

[113] Z. Tu, Z. Su, and P. Devanbu. On the localness of software. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 269–280, 2014.

[114] Twitter. About twitter inc. 2015. [Online; accessed 19-Aug-2015].

[115] G. Uddin and F. Khomh. Automatic summarization of api reviews. In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 159–170, 2017.

[116] G. Uddin and F. Khomh. Opiner: An opinion search and summarization engine for apis. In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 978–983, 2017.

[117] G. Uddin and M. P. Robillard. Resolving api mentions in informal documents. *arXiv preprint arXiv:1709.02396*, 2017.

[118] U. Upadhyay, I. Valera, and M. Gomez-Rodriguez. Uncovering the dynamics of crowdlearning and the value of knowledge. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 61–70, 2017.

[119] H. Valdivia Garcia and E. Shihab. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pages 72–81, 2014.

[120] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov. The sky is not the limit: Multitasking on GitHub projects. In *IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 994–1005, 2016.

[121] B. Vasilescu, A. Capiluppi, and A. Serebrenik. Gender, representation and online participation: A quantitative study of stackoverflow. In *International Conference on Social Informatics (SocInfo)*, pages 332–338, 2012.

[122] B. Vasilescu, V. Filkov, and A. Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 international conference on Social computing (SocialCom)*, pages 188–195, 2013.

[123] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)*, pages 3789–3798, 2015.

[124] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov. How social q&a sites are changing knowledge sharing in open source software communities. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing (CSCW)*, pages 342–354, 2014.

[125] A. J. Viera, J. M. Garrett, et al. Understanding interobserver agreement: the kappa statistic. *Fam Med*, pages 360–363, 2005.

[126] S. Wang, T.-H. Chen, and A. E. Hassan. Understanding the factors for fast answers in technical q&a websites. *Empirical Software Engineering (EMSE)*, pages 1–42, 2017.

[127] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik. Entagrec: An enhanced tag recommendation system for software information sites. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 291–300, 2014.

[128] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik. Entagrec++. *Empirical Software Engineering (EMSE)*, pages 800–832, 2017.

[129] X. Wang, I. Kuzmickaja, K.-J. Stol, P. Abrahamsson, and B. Fitzgerald. Microblogging in open source software development: The case of drupal and twitter. *IEEE software*, pages 72–80, 2013.

[130] X. Wang, D. Lo, J. Jiang, L. Zhang, and H. Mei. Extracting paraphrases of technical terms from noisy parallel software corpora. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 197–200. Association for Computational Linguistics, 2009.

[131] S. Wasserman. *Social network analysis: Methods and applications*. Cambridge university press, 1994.

[132] S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 266–275, 2003.

[133] G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10, 2017.

[134] E. Wong, J. Yang, and L. Tan. AutoComment: Mining question and answer sites for automatic comment generation. In *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 562–567, 2013.

[135] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang. Elblocker: Predicting blocking bugs with ensemble imbalance learning. *Information and Software Technology (IST)*, pages 93–106, 2015.

[136] X. Xia, D. Lo, X. Wang, and B. Zhou. Accurate developer recommendation for bug resolution. In *20th Working Conference on Reverse Engineering (WCRE)*, pages 72–81, 2013.

[137] W. Xu, X. Sun, J. Hu, and B. Li. Repersp: Recommending personalized software projects on github. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 648–652, 2017.

[138] W. Xu, X. Sun, X. Xia, and X. Chen. Scalable relevant project recommendation on github. In *Proceedings of the 9th Asia-Pacific Symposium on Internetware*, page 9, 2017.

[139] A. Zagalsky, C. G. Teshima, D. M. German, M.-A. Storey, and G. Poo-Caamaño. How the r community creates and curates knowledge: a comparative study of stack overflow and mailing lists. In *Proceedings of the 13th International Workshop on Mining Software Repositories (MSR)*, pages 441–451, 2016.

[140] F. Zampetti, L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, and M. Lanza. How developers document pull requests with external references. In *Program Comprehension (ICPC), 2017 IEEE/ACM 25th International Conference on*, pages 23–33. IEEE, 2017.

[141] C. Zhai. Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, pages 1–141, 2008.

[142] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun. Detecting similar repositories on github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 13–23, 2017.

[143] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. 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 (ASE)*, pages 60–71, 2017.

[144] P. Zhou, J. Liu, Z. Yang, and G. Zhou. Scalable tag recommendation for software information sites. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 272–282, 2017.

[145] Y. Zhou, Y. Tong, R. Gu, and H. Gall. Combining text mining and data mining for bug report classification. pages 150–176. Wiley Online Library, 2016.