Empirical Software Engineering (2022) 27: 40
https://doi.org/10.1007/s10664-021-10095-1

A comparative study and analysis of developer communications on Slack and Gitter

Esteban Parra[1] · **Mohammad Alahmadi**[2] · **Ashley Ellis**[1] · **Sonia Haiduc**[1]

Accepted: 2 December 2021 / Published online: 13 January 2022

Abstract

Software developers are often using instant messaging platforms to communicate with each other and other stakeholders. Among these platforms, Gitter has emerged as a popular choice and the messages it contains can reveal important information to researchers studying open source software systems. Uncovering what developers are communicating about through Gitter is an essential first step towards successfully understanding and leveraging this information. In this paper, we first describe the largest manually labeled and curated dataset of Gitter developer messages, named *GitterCom*, obtained by manually analyzing and labeling 10,000 Gitter messages in 10 software projects. We then present a qualitative study to understand the extent to which the categories identified in previous work by Lin et al. (2016) found on Slack through surveys are applicable to developer messages exchanged on Gitter. Further, in an effort to automate the labeling process, we investigate the accuracy of 9 traditional machine learning and deep learning algorithms in predicting the intent of Gitter messages. We found that Decision Trees and Random Forest performed the best, achieving an accuracy of 88%, which is very promising for this multi-class classification task. Finally, we discuss the potential directions for future research enabled by labeled Gitter datasets such as *GitterCom*.

Keywords Gitter · Dataset · Developer communications

Communicated by : Georgios Gousios and Sarah Nadi.

This work is supported in part by the National Science Foundation grant CCF-1526929.

This article belongs to the Topical Collection: *Mining Software Repositories (MSR)*

✉ Esteban Parra
parrarod@cs.fsu.edu

Mohammad Alahmadi
malahmdi@uj.edu.sa

Ashley Ellis
ake17@cs.fsu.edu

Sonia Haiduc
shaiduc@cs.fsu.edu

[1] Florida State University, 600 W College Ave, 32306 Tallahassee, FL, USA

[2] Department of Software Engineering, College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia

1 Introduction

Over time, the increase in the size of software systems has heightened the need for collaboration among developers, and has led to large software development teams, often distributed across multiple locations. Many positive aspects are seen in systems developed collaboratively, such as knowledge sharing, strengthened synergy in teamwork, increased productivity, and increased code quality (Constantino et al. 2020). However, such collaboration is only possible when there is communication among the team members to coordinate their efforts (*e.g.*, share road maps, progress updates, roadblocks or problems, deadlines, and more).

Traditionally, software development teams have used IRC, forums, and mailing lists for communication purposes (Chowdhury and Hindle 2015; Storey et al. 2017; Käfer et al. 2018; Stray et al. 2019). In recent years, modern instant messaging communication platforms such as Gitter[1] and Slack[2] have emerged as a popular alternative for communication among software development teams and other stakeholders (Lin et al. 2016; Storey et al. 2014; Stray et al. 2019). These tools allow for better team collaboration, group awareness, and project coordination by providing a user-friendly way of managing and organizing distributed conversations, facilitating knowledge sharing, and enabling easy access to other team members and their expertise. Moreover, these tools further support distributed agile development teams by integrating relevant information from external software development tools such as GitHub, Travis CI, etc. Storey et al. (2014). Thus, this new wave of communication platforms is bridging the gap between multiple software tools and shaping modern software development activities and practices (Storey et al. 2014).

Given the features and support they offer for software development, many open source projects have adopted Gitter and Slack as their preferred communication means (Ehsan et al. 2020; Käfer et al. 2018). Gitter is a popular tool among open source development teams (Käfer et al. 2018; Sahar et al. 2020; Shi et al. 2021) for a few reasons. First, in Gitter the access to user-generated data is public. In particular, public messages and user-generated content in Gitter are subject to the Creative Commons license: Attribution + Non-Commercial + ShareAlike (BY-NC-SA)[3]. In addition, the messages posted to public Gitter channels are preserved indefinitely in chat room logs, which means that all users can see all messages in a chat room, dating back to when the channel was first created.

Given the popularity of Gitter among open source software communities hosted on GitHub (over 10,000 communities of developers), the history of communications on this platform often contains tens of thousands of message exchanges between the developers of a software system. If leveraged, this history could represent a rich source of documentation for developers looking for specific answers about a system or for on-boarding new developers in a community. At the same time, this information can also be leveraged by researchers wanting to learn more about open source development and the nature of developer communications. Our goal is to enable such endeavors and we do so by introducing the largest manually labeled and curated dataset of Gitter messages, called *GitterCom*. *GitterCom* contains a set of 10,000 messages collected from the Gitter channels of 10 open source software projects and manually labeled based on their purpose, using the categories

[1] https://gitter.im/

[2] https://slack.com/

[3] https://creativecommons.org/licenses/by-nc-sa/3.0/us/

identified in previous work by Lin et al. (2016) through surveys of developers. *GitterCom* is currently the largest manually labeled dataset of developer instant messages, surpassing the previous largest dataset of 500 manually labeled Slack messages (Stray et al. 2019) by two orders of magnitude. We make the data and scripts we used to collect the data available in our replication package (Parra 2021).

This paper builds upon our previously published paper (Parra et al. 2020), which introduced *GitterCom* for the first time and presented a study of the messages it contains with the goal of observing how developers and other stakeholders communicate about software using Gitter in the context of Gitter communities dedicated to the active development of open source software systems on GitHub. In this paper, we further extend this work in a two main ways:

i) We aim to understand whether the categories that Lin et al. (2016) found on Slack through surveys are applicable for messaging data on Gitter, and if so, how prevalent each category is in the two data sources. For this purpose, we compare and contrast the self-reported usage of chat-based communication platforms by developers based on survey responses, as reported by Lin et al. (2016) with the actual usage of these platforms, as revealed by analyzing the messages in *GitterCom*. This study sheds a light on the similarities and differences between the developers' perceived usage and their actual usage of instant messaging platforms like Slack and Gitter. Moreover, we also provide an analysis of message intents across different development communities in *GitterCom*.
ii) We evaluate the use of 9 traditional machine learning and deep learning algorithms for the automatic classification of Gitter developer messages by their intent. Given the continuously increasing amount of data being generated and the time-consuming nature of manual labeling involved in creating this type of dataset, our goal is to facilitate future data collection by automating the classification of messages. Our results show that Decision Trees and Random Forest perform the best, achieving an accuracy of 88%.

The rest of the paper is structured as follows. Section 2 introduces existing work on instant messaging tools and other communication channels used by software development teams. Section 3 outlines the data gathering and labeling procedures we followed for creating *GitterCom*. Section 4 presents a study on the purpose of developer instant messages in the context of open source software development. The study first analyzes the purpose of messages in *GitterCom*, it then compares it to the self-reported usage of instant messaging platforms by developers based on the survey responses reported by Lin et al. (2016), and finally investigates whether traditional machine learning and deep learning algorithms can be successfully used for the automatic classification of developer chat messages. Section 5 discusses potential research directions using *GitterCom*. We discuss threats to validity in Section 6, and finally Section 7 concludes the paper and discusses future work.

2 Related Work

Our work is closely related to the study of developer communications. We divide the related work in two subsections. In the first one, we introduce work done on the analysis of instant messaging communication tools such as Gitter and Slack in software engineering, which is the closest work related to ours. Then, in the second subsection, we present an overview of work related to other communication tools used by developers.

2.1 Instant Messaging Communication Tools in Software Engineering

Earlier work related to our paper has explored the Internet Relay Chat (IRC), the precursor of developer chat communities (Shihab et al. 2009; Elsner and Charniak 2011; Chowdhury and Hindle 2015). Shihab et al. (2009) analyzed the properties of meetings taking place through IRC. In particular, their work looked at the topics, participant size, contributions, and communication styles in these meetings. On the other hand, Chowdhury and Hindle (2015) used machine learning classifiers to extract off-topic discussions, while Elsner and Charniak (2011) introduced a coherence model for separating multiple conversations taking place in IRC channels.

With the rise of modern chat platforms such as Slack and Gitter, recent works have focused on exploring the developer communities using these tools (Anders 2016; Murgia et al. 2016; Storey and Zagalsky 2016; Alkadhi et al. 2017a, b; Lin et al. 2016; Paikari et al. 2018; Käfer et al. 2018; Chatterjee et al. 2019, 2020; Ehsan et al. 2020; Sahar et al. 2020).

Käfer et al. (2018) present the results of an empirical study on the communication tools used in 400 open source software repositories in GitHub. Their results show that mailing lists are being used less and less, with developers favoring modern enterprise chat systems (*i.e.*, Gitter and Slack). Moreover, GitHub Issues, personal e-mail, Gitter, Twitter, and mailing lists were found to be the five most popular communication channels currently used in open source development. Slack was found to be eighth in terms of popularity among all the communication means observed.

Lin et al. (2016) surveyed 53 software developers regarding their use of Slack. They found that developers self-reported using Slack for multiple purposes (*i.e.*, personal benefits, team-wide purposes, and community support), and to support various activities. We make use of the classification of purpose types and their categories and subcategories identified by Lin et al. (2016) in order to manually and then automatically classify developer chat messages in Gitter. Our work differs from theirs in a few ways. First, instead of surveying developers, we use the chat histories to extract and classify actual developer messages. Therefore, while Lin et al. capture the *self-reported usage* of instant messaging platforms by developers, we aim to analyze the *actual usage* of these platforms by looking at messages directly. Using the same categories as Lin et al. (2016) also allows us to compare and contrast our findings to theirs and observe similarities and differences between self-reported and actual developer instant messaging usage. In addition, we also aim to automatically classify the developer messages in order to enable future automatic data collection.

Recent work by Stray et al. (2019) studied a group of 30 developers and their communication through Slack channels at a large software development company. Their analysis involved the open coding of 500 messages, identifying in a broad sense that the messages were related to the following purposes: general information/coordination, general discussions, problem-focused communication, technical communication, and socializing. Their results show that in this company, about half of the messages are related to problem solving (*i.e.*, questions and answers), with very little social talk among the team members. Moreover, by interviewing the team members, the paper shows that language, unbalanced activity, and the excessive use of private messages are the main challenges experienced by the team when using instant messaging tools. While we also perform coding on developer chat messages, our work differs from that of Stray et al. (2019) in several ways. First, we perform a large-scale open coding of 10,000

messages, which is 20 times more than the amount of messages coded by Stray et al. Second, we focus on 10 open source software communities instead of a single closed-source software company. Third, we analyze Gitter instead of Slack. Fourth, our classification is more fine-grained, using the categories identified by Lin et al. (2016).

Some of the recent works on analyzing developer communications in instant messaging platforms have focused on the Q&A conversations taking place on these platforms. Some recent work has also focused on analyzing Gitter communications (Ehsan et al. 2020; Sahar et al. 2020; Shi et al. 2021). More specifically, the work by Ehsan et al. (2020) analyzed 384 Q&A threads in Gitter communities in order to develop a methodology for automatically extracting threads from the chats. Moreover, by analyzing these threads, Ehsan et al. (2020) identified that 80% of them involve users asking how-to questions. The authors then present a set of guidelines that can help users write better questions in order to increase the likelihood of obtaining a response. Work by Chatterjee et al. (2019, 2020) has focused on disentangling and extracting the Q&A conversations within the chats. Their work leverages a supervised model based on a set of features between pairs of chat messages that occur within a window of time of each other. This method expands upon the work by Elsner and Charniak (2011), who presented an initial approach for this problem, by adding additional features that are characteristic to instant messaging platforms (*e.g.*, the use of URLs and channel references). In a recent work, Shi et al. (2021) presents an empirical study on the properties of dialogs/conversation on Gitter by categorizing 749 dialogs/conversations from eight Gitter communities using a set of question categories derived from Stack Overflow by Beyer et al. (2018) and employed social network algorithms and metrics to assess the structure and properties of conversations in chat communication platforms. The paper found that developers tend to discuss topics that are domain-specific to the community. In particular, they found that the discussions on these communities are oriented towards solving problems such as API usages and errors. Moreover, among the issues and errors discussed, developers discuss more 'unwanted behavior' and 'do not work' errors than issues relating to reliability, performance, or test/build failures. The paper also found that the social network of developers within the studied communities can be categorized into three types of networks (polaris, constellation, and galaxy). Lastly, the paper identified six dialog interaction patterns in the live chat communities. Work by Sahar et al. (2020) focused on analyzing how developers discuss issue reports within Gitter communities devoted to open source systems. Their work extracted and analyzed references to issue reports in the Gitter channels of 24 active open source development systems. The empirical analysis of these issue discussions shows that end users referenced the majority of the issue reports. Moreover, the two most common reasons these issues are brought up in Gitter are i) to reference an issue than contains additional information on a problem or topic being discussed and ii) to inform on the opening, closure, or comments to the issues themselves. Moreover, their results indicate that Gitter might be used by developers and users to revive and facilitate resolution of issues that have not been addressed in a long time in the project's issue tracking system.

A closely related work to ours concerns the identification of rationale in development chat messages (Alkadhi et al. 2017a, b). Rationale arguments are any messages by the developers that contain information justifying the decisions made throughout the software life cycle. The work by Alkadhi et al. (2017b) presents an exploratory study on the presence of rationale in the chat development messages of three development teams made up of students working on a multi-project capstone course. Their findings show the presence of rationale, as well as the usefulness of SVM and Naïve Bayes classifiers toward the automatic identification and classification of messages containing rationale information.

In subsequent work, Alkadhi et al. (2017a) introduce REACT, an approach for developers to explicitly record the rationale in messages via manual annotation using either in-line annotations or Slack reactions, capturing five rationale elements: issues, alternatives, pro-arguments, con-arguments, and decisions. Our work is different from that by Alkadhi et al. as we focus on the identification of the *purpose* of developer communication on the chat platforms of *active open source projects*, rather than the identification of rationale in student projects.

To the best of our knowledge, our work: i) presents the largest dataset of developer chat messages annotated by their purpose, containing 10,000 manually labeled Gitter messages; ii) is the first to compare the self-reported, general usage of instant messaging platforms to the actual usage of these platforms by developers; iii) is the first to show the effectiveness of multi-class machine learning classification towards the automatic identification of the *purpose* of open source developer messages in modern chat platforms. The works by Ehsan et al. (2020), Sahar et al. (2020), and Shi et al. (2021) also use Gitter as the target of their research. However, Ehsan et al. (2020) focused on identifying and analyzing conversation threads, while our work analyzes and automatically identifies the purpose of individual messages, Sahar et al. (2020) analyzes a subset of Gitter messages that are encompassed within the Dev-Ops and costumer support categories in our study, and Shi et al. (2021) focus on analyzing the structure of the communities and the information at a conversation level.

2.2 Other Communication Tools in Software Engineering

One of the most studied avenues of developer communications are Q&A forums, such as Stack Overflow. Stack Overflow allows developers from across the globe to communicate with each other by posting question and answers to topics ranging across different programming languages and software development topics (Allamanis et al. 2013; Linares-Vasquez et al. 2013).

Stack Overflow has been used to support software developers by mining API descriptions and examples (Keivanloo et al. 2014), generating source code comments (Vassallo et al. 2014), extracting code snippets (Subramanian and Holmes 2013), helping with bug triaging (Sajedi Badashian et al. 2016), summarizing answers to technical questions (Xu et al. 2017), etc. The popularity and vast amounts of communications among developers in Stack Overflow has also lead to research into using the information present in these forums to build a software-specific word similarity database similar to WordNet (Tian et al. 2014). Moreover, there has been research towards analyzing and improving the developer interactions with Q&A forums by analyzing the characteristics of developers (Novielli et al. 2014, 2015), exploring the way developers write answers and questions (Treude et al. 2011; Nasehi et al. 2012; Arora et al. 2015; Ponzanelli et al. 2014), tagging questions (Rekha et al. 2014), and mentoring developers on how to best write questions in Q&A forums (Ford et al. 2018).

Open source developers also use Twitter to communicate information related to the software development. Work by Fang et al. (2020) analyzed Twitter messages by repository owners and found that they do not engage in work discussion or answering others people's questions. Instead, they mostly use Twitter to share work-related information, development updates, and advertise their own work. Guzman et al. (2017) presents ALERTme, an approach to classify Twitter messages that may be relevant for developers. These tweets can include: problem reports, improvement suggestions, or user needs.

3 GitterCom

In this section we present *GitterCom*, a manually curated dataset containing 10,000 messages collected from 10 open source software development Gitter communities (1,000 messages per community). Each message in the dataset was manually labeled with information about the purpose of the communication it expresses, based on the categories identified by Lin et al. (2016). *GitterCom* is made available online in two formats: CSV and Microsoft Excel Open XML Spreadsheet (XLSX) file formats[4]. In both formats, each line contains the information for a single message and consists of seven information fields, separated by comma and using quotes as the text delimiter in the CSV format. In particular, each line contains: (i) the channel/system the message belongs to, (ii) a unique message ID, (iii) the date and time at which the message was posted, (iv) the author of the message, (v) the content of the message in plain text, (vi) the corresponding high-level purpose (manual label), (vii) the purpose category (manual label), and (viii) the purpose subcategory (manual label).

The following subsections describe the categories used to classify the developer messages and the procedure we followed to collect the data and obtain the final *GitterCom* dataset.

3.1 Message Categorization

Recent works on the use of instant-messaging tools have derived categorizations for messages exchanged by developers. First, after conducting a survey with software developers who adopted Slack, Lin et al. (2016) identified that developers self-report using chat rooms for three main purpose types (*i.e.*, personal benefits, team-wide purposes, and community support). Moreover, they found that within each purpose type, developers use instant messaging communication tools to support different tasks and activities (which can be seen as purpose categories and subcategories for these main types). Second, Stray et al. (2019) also derived a set of categories for developer messages exchanged in Slack by analyzing 500 Slack messages in a large software development company.

We decided to use the purpose types, categories, and subcategories of messages identified by Lin et al. (2016) in our work for several reasons. First, there is evidence that categories derived from one type of software developer communications are applicable to describe the communications of developers in a different communication medium. In the latest such example, Shi et al. (2021) used categories derived from StackOverflow to classify conversations in Gitter. Since both Slack and Gitter are chat-based platforms that allow developers to exchange messages in chat rooms, we believe the type of information exchanged within these communities is likely to be even more similar, since the purpose of the two platforms is the same. Slack and Gitter are fundamentally similar tools serving the same goal and developers interact the same way in these platforms, by posting and replying to instant messages. We believe the kind of information exchanged within these communities by developers is not impacted by the specific tool they use (*e.g.*,, just like developers use different bug tracking tools to serve the same purpose), given that the nature of communicating itself is the same for both Slack and Gitter: instant messages in community channels. Therefore, we considered the classification of

[4] https://figshare.com/s/576d328da4a5b50ea155

Table 1 Hierarchy of purpose types, categories, and subcategories identified by Lin et al. (2016) and used in our paper

Purpose Types	Purpose Category	Purpose Subcategories
Personal benefits	Discovery and news aggregation	Interesting/relevant blogs
	Networking and social activities	Similar interests
		Similar jobs
	Fun	Gaming
		Sharing gifs and memes
Team-wide purposes	Communication	Communication with teammates
		Communication with stakeholders
		Non-work topics
	Team collaboration	Team management
		File and code sharing
	Customer support	Bugs
		Troubleshooting
		How-to
	Dev-Ops	Development operation notifications
		Software deployments
		Team Q&A
Community support	Communities of practice	Keep up with specific frameworks/communities
		Bouncing ideas off of other people in the community
		Learning about new tools and frameworks

Lin et al. (2016) to be a good starting point, while also keeping the door open to adding new categories or redefining the categories altogether should we find it necessary during our manual labeling. However, we found that the categorization provided by Lin et al. (2016) was very comprehensive and accurately reflected the kind of information we identified in messages during our labeling. Therefore, we found that no modifications or additions were needed to accurately reflect the developer communications in GitterCom, which confirms the fact that the fundamental purposes of communication between the two tools are very similar.

Second, we decided to use the categorization used by Lin et al. (2016) instead of that provided by Stray et al. (2019) because the latter was based on messages from developers working at a single software company and could, thus, be less generalizable to other software companies or open source software systems, which are our main focus. In addition, we performed a close inspection of the categories derived by Stray et al. (2019), and found that they are a subset of the finer-grained categorization identified by Lin et al. (2016).

We additionally explored the use of Latent Dirichlet Allocation (LDA), a popular automated topic modeling technique to derive topics that could be used to classify the messages within *GitterCom*. However, we found the resulting topics to be inadequate for describing the higher-level purpose of the messages and inferior to those already proposed in previous work (Lin et al. 2016; Stray et al. 2019). The detailed results of the LDA modeling can be found in our replication package.

For the rest of the paper, we use the purpose types, categories, and subcategories identified by Lin et al. (2016) for classifying the messages in our dataset. We present the

hierarchy of these purpose types, categories, and subcategories of developer messages in Table 1. A brief description of each is presented below, based on Lin et al. (2016):

- **Personal benefits**: Messages in which the developer's main purpose is to fulfill personal needs. Messages within this purpose type can be further divided into three categories:
  - *Discovery and aggregation of news and information*, where developers post reliable, interesting, and relevant blogs or other sources of information.
  - Messages supporting *networking and social activities* with other developers who share similar interests or jobs.
  - *Fun* messages sharing gifs and memes or for participating in gaming activities.
- **Team-wide purposes**: Messages aimed towards carrying out software development activities related to the system being developed. Messages within this purpose type can be further divided into four categories:
  - *Communication*, representing messages in which developers engage in activities such as communication with teammates (*e.g.*, members of a distributed team) during meetings and note taking, communication with other stakeholders, or discussing non-work topics.
  - *Team collaboration* through team management, file and code sharing.
  - *Dev-Ops*, including messages that communicate updates regarding the status of the system (*e.g.*, development operation notifications about recent changes to the system, commits, bug fixes, pushes to the repository, merges), software deployments, and team Q&As.
  - *Customer support* messages, which can incur when assisting new users of the system on how to perform certain tasks, troubleshoot errors, identify bugs.
- **Community support** messages in which developers participate in communities of practice or special interest groups, where they can keep up with specific frameworks/communities, bounce ideas off of other people in the community, or learn about new tools and frameworks for developing applications.

3.2 Data Collection

In order to create GitterCom, we first looked at Gitter communities of active open source software systems. A Gitter community can have multiple channels devoted to different conversation topics. These channels are chat rooms can be about anything and do not have to map directly to something in GitHub. For instance, Gitter's own community[5] has one room devoted to communication between developers and one room for customer support.

The communities in our dataset have between 1 and 89 channels, with a median of 2 channels per community. Similarly to Shi et al. (2021), after inspecting the channels within the communities, we found that, at the time of data collection, the majority of the historical message exchanges and user activity in Gitter communities of open source projects have taken place within their *main channel*.

[5] https://gitter.im/gitterHQ/home

Table 2 User count per room within the ImageJ Gitter community

Channel	Users	Description
imagej/imagej-omero	19	Server- and client-side communication between ImageJ and OMERO
imagej/imagej	273	Open source scientific multidimensional image processing
imagej/imagej-ops	67	ImageJ Ops: "Write once, run anywhere" image processing
imagej/imagej-server	12	
imagej/imagej-updater	14	
imagej/hackathon-Ostrava-2019	29	
imagej/openmpi-parallelization	3	Developing OpenMPI-aware plugins for Fiji
imagej/hackathon-dresden-2019	32	
imagej/imagej.github.io	43	This is where we discuss the ImageJ wiki (https://imagej.net/)
imagej/pyimagej	15	Developer discussion for PyImageJ

Table 3 Some descriptions of the Gitter channels explored during the data collection

Channel	Description
ceylon/dev	Welcome to the developer's discussion channel for the Ceylon programming language. This is where we yell at each other when things are broken again; while it's a public channel, we consider it more internal than ceylon/user, and won't control our speech as much here. Enter with caution :
alcatraz/Alcatraz	Xcode package manager
airbnb/caravel	DEPRECATED CHANNEL - GOTO https://gitter.im/airbnb/superset
dev-ua/clojure	Do you<3 Clojure? Enjoys ClojureScript? Welcome to our community! FAQ: https://gist.github.com/listochkin/c81c198a2b-7b044a0dc5
ethereum/light-client	https://github.com/zsfelfoldi/go-ethereum/wiki/Geth-Light-Client

Furthermore, we performed an analysis of the communities by gathering data about the channel descriptions and the number of users within the channels in each community as of July 2021. Table 2 presents details regarding the channels in the ImageJ community as a representative example. As seen in Table 2, among the 10 channels in the ImageJ community, the main channel (*i.e.*, imagej/imagej) is the one with the most active members.

Each channel in Gitter can have an associated description provided by the administrators of the channel. Some examples of descriptions are shown in Tables 2 and 3. After obtaining the descriptions for the 653 channels associated with the 139 communities in our raw data, we found that 218 of the channels do not have a description and 9 of the channels have descriptions written in languages other than English. We excluded these 227 channels from our analysis and performed an analysis of the descriptions of the remaining 426 channels. For this analysis, we inspected the descriptions and derive insights on the information that they provide to developers looking to join or participate in the communities.

The descriptions of Gitter channels that were not left blank were observed to have various lengths, starting at a single word or url, and going all the way up to 53 words which provided a thorough outline of the community and what type of communication was expected within the channel. We also found four instances of channel descriptions that do

not provide any meaningful information regarding the community (*e.g.*, "intentionally left blank", "this is the topic"). However, the vast majority of the channels' descriptions provide at least a description of the system/programming language the channel is devoted to. A large percentage of the channels opt for shorter descriptions, with 70.66% of them having descriptions shorter than 10 words, while 22.30% of them have descriptions between 10 and 20 words, and only a 7.04% of the channels have longer descriptions of 20 words or more.

During our analysis, we found a total of ten channels which indicate in their descriptions that they are meant for internal communication (*i.e.*, only for developers of the system). We found that developers will include several key terms in the channel description to indicate that a channel is intended for internal communication (*e.g.*, "internal", "for committers", "coordinate development", and "contributors"). Moreover, we found 117 of the descriptions contain urls. We inspected these urls and identified that they serve one of three main purposes: providing a quick access to a frequently asked question (FAQ) page regarding the channel{11}, directing the users to either the website, documentation, or the GitHub repository of the system the channel is associated with{66}, or redirecting the user to either an alternative channel or a different communication platform (*e.g.*, Discord){42}. Two of the descriptions contained urls to both the website and to an alternative communication channels/platforms.

Given the message and user distribution within rooms, when selecting messages for our analysis, we focused on messages posted in the main rooms of Gitter communities. To collect the data for the *GitterCom* dataset, we first gathered the list of all the Gitter communities highlighted in Gitter's Explore interface[6] on April 1, 2019 (the day of our data collection). For new users, such as the user we created when collecting our dataset, the "Explore" page is populated by querying up to 50 rooms/communities per tag based on a predetermined set of 25 tags (*e.g.*, "Mobile", "iOS", etc.), which are hardcoded in the source code of Gitter. Gitter's "Explore" page lists the top rooms by number of users for each tag, sorted in descending order and only lists rooms that contain at least one user. Thus, our initial step on data collection resulted in a list of 139 Gitter communities. We then used Gitter's API[7] via a custom python script to extract all of the messages in the main rooms of these communities and their corresponding metadata, from their inception until April 1, 2019. This resulted in a set of 2,939,335 messages across all 139 communities. Our data gathering procedure is similar to the one followed by Sahar et al. (2020). We then excluded 3 communities in which the conversations were not in English. Afterwards, to facilitate the labeling process, we ran a custom script in Java to convert the messages from the JSON format provided by Gitter's API to CSV format.

The data collection scripts, instructions on their usage, and statistics on message and user distribution in the rooms and communities, are found in our replication package (Parra 2020).

After exploring the collected channels we noticed that the they vary in several ways: (i) *by membership* - the channels contain between 101 and 17,000 members per channel, (ii) *by level of activity*, with the least active channel containing only 21 messages, and the most active one containing over 423,000 messages since its inception, and (iii) *by type*, as channels can be designated for the development of a particular software system, where the developers communicate with each other and with the system's users, or

[6] https://gitter.im/explore

[7] https://developer.gitter.im

Table 4 Gitter communities included in GitterCom

Community	Users	Messages	Application domain
Marionette (mar 2020)	3014	181108	Javascript framework
jspm (jsp 2020)	1103	27245	Package manager
scikit-learn (sci 2020)	3188	9844	Machine Learning
Xenko3d (xen 2020)	103	2890	Game engine
FreezingMoon (fre 2020)	109	207925	Video game
UIkit (uik 2020)	2155	41265	Front-end framework
jHipster (jhi 2020)	2575	39418	Application generator
Cucumber (cuc 2020)	337	2030	Testing framework
Imagej (ima 2020)	209	8149	Image processing
TheHolyWaffle (thw 2020)	196	15046	VoIP communication

made for building communities of practice in which the members' discussion revolves around particular topics, frameworks, or programming languages, but does not involve discussion about the active development of a system.

Since our focus in this work is on developer communications related to the development of open source software systems, we were interested in Gitter channels associated with active open source systems. To determine this, we verified that the communities are linked to a GitHub repository where commits have been made within the past year and also made sure that the messages in the Gitter channel contained several conversations within the last month at the time of coding that were devoted to discussing the development of the system. We considered only GitHub since at the time of data collection, Gitter channels only had the ability to directly link to GitHub repositories and not to repositories hosted on different services such as GitLab.

After inspecting the 139 communities, we determined that the presence of at least 1,000 messages in the main channel in the past year is a good indicator of an active community, with users engaging in various conversations. Moreover, due to the expected human effort required for labeling the data, we chose to focus on 10 communities as it was an achievable goal.

We randomly selected channels out of the 139 we found until we had 10 that met all of the following selection criteria: (i) they are linked to an active GitHub repository where commits have been made within the past year, (ii) they are used as a communication tool for the active development of an open-source software system, (iii) users have been active in the channel in the past year, (iv) they have at least 1,000 messages since inception, and (v) they cover different application domains. Table 4 shows the details of the selected systems/channels.

From each of the 10 channels, we then collected the 1,000 most recent consecutive messages up to April 1, 2019, for a total of 10,000 messages across the 10 channels. We choose to label consecutive messages rather than randomly selected messages to ensure that the dataset accurately represents the conversations that developers engage in when communicating using instant messaging platforms. Moreover, we used the most recent messages as they would best represent recent communication and usage of the platform.

Two of the authors then carried out a coding procedure to label these messages, using the purpose types, categories, and subcategories identified by Lin et al. (2016) (see Table 1) as labels. More specifically, each message was assigned a purpose type, a

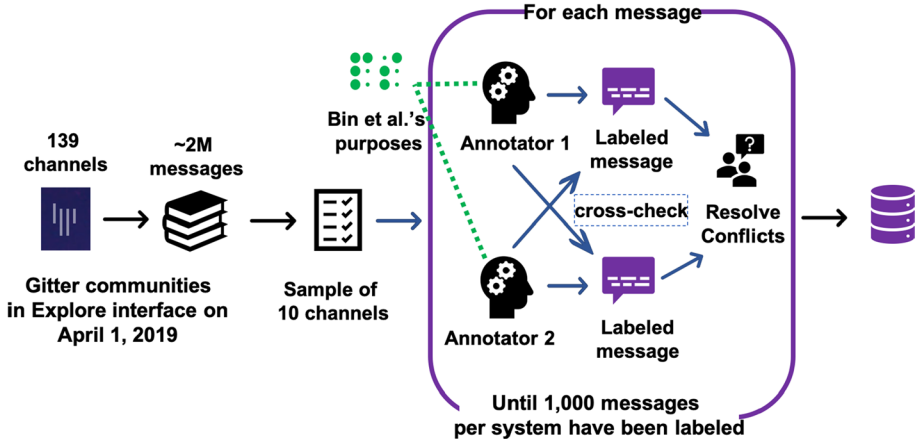

Fig. 1 Manual annotation procedure

category describing the main purpose of the message, and a subcategory describing the specific activity the message relates to.

The procedure employed when labeling the messages was as follows (See Fig. 1). For each of the 10 channels, the 1,000 most recent consecutive messages were independently classified by two of the authors into the purpose types, categories, and subcategories outlined in Table 1. Messages within the "communication" category are those in which the developers communicate among themselves or with other stakeholders, but the messages are not directly associated with a particular software/system. For example, messages among developers regarding making Scala objects more like Java objects using lambdas would be classified as "communication" as they involve communication with teammates but they are not related to a particular system. Other examples of messages that are classified as "communication" include messages discussing personal projects not related to the system being developed, or discussions about updating the user profile picture on Gitter (classified as "non-work").

Lastly, if a message did not provide any meaningful information by itself (*e.g.*, a single emoji, "k", "cool", empty messages), it was classified as "Uninformative". After the individual coding, the two authors got together, discussed and resolved any coding conflicts. Then, the messages for which a classification of "Uninformative" was agreed upon were discarded and replaced by an equal number of messages from the same channel. Then, the coding process was applied on these new messages. This procedure was repeated until 1,000 messages were obtained for each channel, all having a label other than "Uninformative". In total, 1,072 messages across all 10 channels were labeled as "Uninformative" during this process.

In addition to the content of the messages, we used the list of contributors to the system's repository to better classify the messages in cases where the content of the message might be insufficient to determine a category. One example were questions which could be interpreted as either a customer asking about the system (Customer Support) or a developer of the system asking about a part of the system they are unfamiliar with (Team Q&A). In this particular case, if a question was made by a contributor to the system it was classified as Team Q&A, and Customer Support otherwise. It is also

important to note that for these two categories, the question, answer(s), and any consequent clarification are assigned the same category.

The manual coding procedure took the two authors about 100 hours per person to complete (200 hours total) and spread across three weeks with a Cohen-kappa agreement of 0.89. The annotators resolved a total of 835 conflicts when labeling the messages. During the discussion, the majority of conflicts were messages classified as "customer support" but were in fact instances of the "Dev-Ops" category, due to the user asking the question being a developer rather than a user of the system. Other instances of disagreement included messages labeled as "communication" when they were a better fit for other categories (e.g., fun, team management, team Q&A). Although we were open to creating new categories during the labeling process, we found the categories by Lin et al. (2016) to be appropriate to describe the purpose of all the messages encountered during labeling.

After completing the manual labeling, we obtained *GitterCom*, a dataset comprised of 10,000 Gitter messages, 1,000 per Gitter channel, classified according to their purpose. The dataset can be found in our replication package (Parra 2021). Some samples of messages and their categories are presented in Table 5.

4 Study on the Purpose of Developer Instant Messages in Open Source Software Development

Based on the *GitterCom* dataset we collected, we perform an empirical study aimed at gaining insights into how developers use instant messaging communication tools in the context of open source software development by analyzing the *purpose* of the messages they exchange. In addition, we are also interested in comparing the actual developer usage of instant messaging platforms derived from these messages to the self-perceived usage that developers report. In other words, we want to compare what developers *believe* they are using instant messaging platforms for versus what they are *actually using* them for.

Since one of our goals in creating *GitterCom* was to enable other researchers to perform analyzes and obtain insights into developer communications using instant messaging platforms, we also wanted to see if we could further enable researchers to scale up this kind of analysis. While building *GitterCom*, we experienced first-hand the immense manual effort involved in labeling historical chat data. In order to reduce this manual effort for potential future studies of developer communications by other researchers, we investigate the use of machine learning algorithms to automatically classify developer messages. Moreover, we believe this kind of automatic techniques could potentially help also the developer communities, as developers may benefit from being presented with the messages that correspond to a particular purpose and category based on their current information needs, so they spend less time going through potentially irrelevant information.

With all this in mind, we formulated the following research questions that we aim to address in this study:

RQ1. **What is the purpose of the instant messages written by developers when using chat platforms in the context of open source software development?**
RQ2. **How does the self-perceived usage of instant messaging platforms reported by developers compare to their actual usage of these platforms?**
RQ3. **Can machine learning classifiers be used to automatically identify the purpose of messages exchanged in developers' instant messaging communications?**

Table 5 Representative examples of messages in our dataset

Channel	Message	Category	Subcategory
Cucumber	@aslakhellesoy I think I'm making some good headway. I also noticed there's a possibility for duplicate feature paths, I'm adding a unit test for that and a fix. I'm planning on removing the CucumberFeature stuff and just create a TreeMap with key(featurepath) and value(feature).	Dev-Ops	Development operation notifications
ImageJ	(See [this blog post](http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html) for details.)	Discovery and aggregate news and information	Interesting and relevant blogs
Marionette JS	going to do a release in like 5 min	Dev-Ops	Software deployments
Scikit-learn	took me only 3 days to catch up with 7 days of sklearn notifications....	Communication	Communication with teammates
JSPM	Personally I think authors should not care about 1000 ways to package their project. They just need to declare what module system they use (at the same time declaring module system they expect of dependencies) and publish only in that module system.	Participation in communities of practice	Bouncing Ideas off of others in the community

4.1 Methodology

To answer **RQ1**, we performed a descriptive statistics analysis on the purpose categories of the developer messages in *GitterCom*. In particular, we investigated the percentage of messages in each purpose category across the entire dataset, as well as the number of messages in each category, in each of the systems in *GitterCom*.

To answer **RQ2**, we compare the results obtained in RQ1, which reflect the actual purposes that developers use instant messaging platforms for, with the self-perceived usage of these platforms reported by developers in the survey responses collected by Lin et al. (2016). This allows us to understand whether the categories of Lin et al. (2016) found on Slack through surveys are applicable to messaging data on Gitter, and if so, how prevalent each category is in the two data sources. Since we used the same categories as Lin et al. (2016), we can directly compare and contrast what developers *believe* they use instant messaging for with what they *actually* use these tools for. Although we contrast results derived from two different chat-based instant messaging platforms, namely, Slack, used by Lin et al. (2016) and Gitter, used to collect *GitterCom*, the two tools offer extremely similar functionality and have the same goal, and the instant messages collected, as well as the survey-based opinions do not depend on any specific features of the two platforms.

To answer **RQ3**, we explore the use of several supervised machine learning algorithms, both traditional and deep learning-based, to automatically determine the categories of messages exchanged by developers. We first describe the methodology we used to train these algorithms, followed by a description of the algorithms themselves. We also release the complete scripts containing the implementation we used for these algorithms in our replication package (Parra 2021).

As later described in the results of RQ1, we noticed that messages in *GitterCom* have mostly "team-wide" purposes. Due to the low frequency of messages for "personal benefit" and "community support" purposes, it is unfeasible for machine learning classifiers to learn meaningful boundaries for each of these categories. Therefore, we aggregated these messages into a single category called "Other purposes" for the scope of this research question. Therefore, the classes considered for this research question are the four categories under the "team-wide" purpose type in Table 1 and the fifth category, "Other purposes".

We train and evaluate the performance of the various multi-class machine learning classifiers on predicting the category of each message among the five possible ones using the ground truth in *GitterCom*. We also compare the machine learning algorithms against a baseline represented by a random classifier. For the random baseline classifier, we used Python's stratified DummyClassifier. The baseline classifier predicts classes for the samples randomly while respecting the training set's class distribution.

For our analysis, we first removed punctuation, numbers, special characters and common English stopwords from the data, afterwards, we performed 10-fold cross-validation using the standard implementation of each algorithm in the scikit-learn machine learning library for Python. For the LSTM and CNN algorithms we use the Keras[8] framework. We first executed the classifiers without any special parameter tuning. This allowed us to first compare the classification algorithms in their default state. Since additional tuning can further improve performance, we then selected the top two classifiers in terms of accuracy and

[8] http://keras.io/

performed hyper-parameter tuning using 10-fold cross-validation to find the best parameters for each model.

To evaluate the classifiers, we use standard evaluation metrics which have been previously used in machine learning applications on software engineering problems (Seiffert et al. 2014; Panichella et al. 2015). Since we are interested in the overall performance of the classifiers, we use *accuracy* as our goal metric. However, we also report *precision* and *recall* for completeness. The definitions of these metrics are provided below.

Accuracy is the ratio of properly classified samples out of the total number of samples in the dataset and its formula is: $Accuracy = \frac{tp+tn}{total-number-of-samples}$, where *tp* is the number of true positives and *tn* is the number of true negatives.

Recall is the computed as the number of true positives over the number of true positives plus the number of false negatives. It measures the ability of the classifier to identify the positive samples. Its formula is: $Recall = \frac{tp}{tp+fn}$

Precision is the ratio of true positives over the number of true positives plus the number of false positives. It measures the ability of the classifier not to label as positive a sample that is negative. Its formula is: $Precision = \frac{tp}{tp+fp}$

We performed hyperparameter tuning for each of these classifiers based on accuracy. During the hyperparameter tuning each classifier is trained and evaluated via 10-fold cross validation with various hyperparameter configurations using Python's grid search[9]. For LSTM and CNN, however, we performed random search, as it has been shown to be more efficient than grid search for hyperparameter optimization in neural networks (Bergstra and Bengio 2012). The scripts used for hyperparameter tuning can be found in our replication package (Parra 2021).

In our evaluation of the machine learning algorithms we used the best performing configurations for each classifier resulting from the hyperparameter tuning.

In the following subsections we now describe the supervised machine learning classification algorithms we used in our study. Supervised classification algorithms aim at producing a learning model from a labeled training set. In our case, we have a multi-class classification problem in which each message can be classified using one of 5 purpose categories. Several algorithms have been proposed to solve this type of problem (Aly 2005). In general, machine learning algorithms address the multi-class classification problem either by a natural extension of the algorithms designed for binary classification or by converting the multi-class classification problem into a set of binary classification problems that are efficiently solved using binary classifiers.

In our work, we explore the following 9 multi-class classifiers, using the standard term frequency-inverse document frequency (TF-IDF) word vectors as features, as done in previous software engineering research (Poché 2017): Stochastic Gradient Descent (SGD) (Bottou 2010), Decision Trees (DT) (Safavian and Landgrebe 1991), Random Forest (RF) (Breiman 2001), k-Nearest Neighbor (kNN) (Khan et al. 2010), AdaBoost (Hastie et al. 2009), Naive Bayes (NB) (McCallum and Nigam 1998), Support Vector Machines (SVM) (Cortes and Vapnik 1995), and two neural network architectures: a Recurrent Neural Network composed of Long Short-term Memory units (LSTM) (Gers et al. 2002), and a Convolutional Neural Network (CNN) (Kim 2014). We present a brief description of each classification algorithm below. For a more in-depth discussion of each algorithm, we direct the interested reader to the papers that initially introduced them and for a side-by-side

[9] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

comparison, we recommend the reviews of these algorithms (Singh et al. 2016; Khan et al. 2010).

4.1.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a simple, yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has more recently received a considerable amount of attention in the context of large-scale learning as it has been shown to have high performance for large-scale problems (Bottou 2010).

4.1.2 Decision Trees (DT)

A Decision Tree is a classification algorithm that learns simple decision rules inferred from the data features. The DT rebuilds the manual categorization of data by constructing well-defined true/false-queries in the form of a tree structure, hence the name. In a DT structure, leaves represent the corresponding category of documents and branches represent conjunctions of features that lead to those categories (Safavian and Landgrebe 1991).

4.1.3 Random Forest (RF)

Random Forest is part of a set of machine learning classifiers known as ensemble classifiers. Ensemble classifiers combine the predictions of several base classifiers built with a given learning algorithm in order to improve the generalizability and robustness over a single classifier (Breiman 2001).

An RF is a combination of multiple independent decision trees, where each tree is built from a sample drawn with replacement from the training set. When splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree). However, due to averaging, its variance also decreases. The amount decreased is usually more than enough to compensate for the increase in bias, hence yielding an overall better model (Breiman 2001).

4.1.4 AdaBoost

Like RF, AdaBoost is an ensemble classifier. AdaBoost is an iterative procedure that combines many weak classifiers. Starting with an unweighted training set, AdaBoost builds an initial classifier to produce a classification. If a training data point is misclassified, the weight of that training data point is increased (boosted). Then, a new classifier is built using the new weights. The classification and boosting procedure is repeated to produce multiple classifiers. Lastly, the final classifier is defined as the linear combination of the classifiers from each stage. AdaBoost has been shown to minimize the exponential loss, making it highly competitive in terms of misclassification error rate (Hastie et al. 2009).

4.1.5 k-Nearest Neighbor (kNN)

K-Nearest Neighbor aims to find the k training samples closest in distance to a new element and then predicts the label of this new element from its k-nearest points. The distance can generally be any similarity function. Despite its simplicity, kNN is often successful in classification situations where the decision boundary is very irregular (Khan et al. 2010).

4.1.6 Naïve Bayes (NB)

Naïve Bayes is an efficient linear probabilistic classifier that uses Bayes' theorem to identify strong (naive) assumptions between features. NB assumes that all of the features in a given class are conditionally independent of each other Russell and Norvig (1995).

In the context of text classification, the multinomial NB model captures word frequency information in the documents using a unigram language model with integer word counts. Each document is then typically represented as a vector of integer or real number attributes, which indicate the importance of words in the document (McCallum and Nigam 1998).

4.1.7 Support Vector Machine (SVM)

Support Vector Machine is a supervised machine learning algorithm used for binary classification, regression analysis, and other tasks, like outlier detection in multidimensional data spaces. An SVM seeks to find a hyperplane, which separates two classes of samples by the maximal margin, in a high dimensional feature space. It can be mathematically proven that the hyperplane parameters depend only on a subset of the training samples, which are called support vectors. To classify a test sample, it is first projected to the feature space and then assigned a class based on which side of the hyperplane it lies on Cortes and Vapnik (1995).

4.1.8 Long Short-Term Memory (LSTM)

Deep learning classification algorithms that automatically learn compositional representations of documents have been successfully applied in the fields of speech recognition, machine translation, text information retrieval, and natural language processing (Mikolov et al. 2011; Deng 2014; West 2000). The success of deep learning in the NLP field and other software engineering tasks encouraged us to apply this type of approach for classifying messages in software development chat communications. We used the pre-trained Glove word embeddings (Pennington et al. 2014). We also evaluated the neural networks using the software engineering word embeddings, in the word embedding layer (Efstathiou et al. 2018).

An LSTM is a type of neural network that uses enriched gating units to avoid the scaling effect, by ensuring that the scaling factor is fixed to one. In an LSTM unit, the memory block contains one or more memory cells and three adaptive, multiplicative gating units shared by all cells in the block. Each memory cell has, at its core, a recurrently self-connected linear unit called the Constant Error Carousel (CEC). The CEC provides short-term

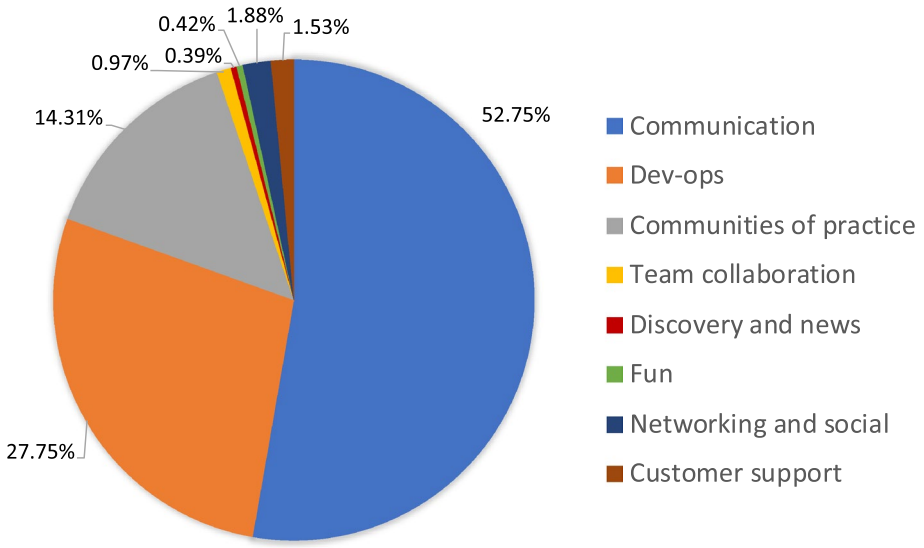

Fig. 2 Distribution of messages by category

memory storage for extended periods by recirculating activation and error signals indefinitely (Gers et al. 2002).

In this paper, we use an LSTM architecture composed of five layers. The first layer is a trainable embedding layer that maps the input text into low-dimensional word vectors, with two LSTM layers, followed by two fully connected layers with dropout in between.

4.1.9 Convolutional Neural Network (CNN)

A Convolutional Neural Network is a type of neural network that uses a combination of layers that apply a convolution operation to local features to extract those that encode semantic features of words in their dimensions (Kim 2014).

In this paper, we use a CNN architecture composed of four layers. The first layer is a trainable embedding layer that maps the input text into low-dimensional word vectors, with two convolutional layers with dropout, followed by one fully connected layer.

4.1.10 Over-/Under-sampling

In addition to the machine learning algorithms, we explore the use of under-sampling and over-sampling algorithms to mitigate the effect of data imbalances and non-linear separability in our dataset. More specifically, we use SMOTE, Tomek links, and the Neighborhood Cleaning Rule. We use the Python implementation of these algorithms in the imbalanced-learn library[10].

[10] https://imbalanced-learn.readthedocs.io/en/stable/api.html

Table 6 Distribution of messages per purpose category

Category	cuc	fre	ima	jhi	jsp	mar	sci	thw	uik	xen	Overall
Communication	325	794	490	446	635	695	506	583	321	480	5275
Customer support	442	0	150	239	0	0	4	145	451	0	1431
Dev-Ops	198	183	308	269	305	235	464	240	190	383	2775
Discovery and news	13	1	10	9	7	2	3	15	5	32	97
Fun	0	2	0	0	0	39	0	0	1	0	42
Networking and social activities	0	0	1	3	0	3	0	0	0	32	39
Participation in communities of practice	4	2	9	15	21	13	13	10	12	54	153
Team collaboration	18	18	32	19	32	13	10	7	20	19	188

SMOTE is an over-sampling approach in which "synthetic" examples of the minority classes are created by interpolating existing minority samples rather than by over-sampling with replacement. SMOTE focuses on generating new minority class instances near borderlines with SVM to help establish a boundary between classes (Chawla et al. 2002).

Tomek Links is an under-sampling approach to remove the majority samples involved in a Tomek link and increases the linear separability of the classes since the retained boundary samples are better chosen (*i.e.*, close to the decision boundary). A Tomek link exists when two samples from distinct classes are the nearest neighbors of each other. Samples that create a Tomek link have been shown to be either borderline or noisy (Tomek 1976).

The Neighborhood Cleaning Rule (NCL) is an under-sampling approach that aims to identify noisy and redundant data points to be removed using the edited nearest neighbor rule. In particular, NCL removes points that are misclassified by their 3-nearest neighbor. Secondly, the neighbors of each positive sample are found and the ones belonging to the majority class are removed. The NCL aims to improve the classification of underrepresented classes while retaining the ability to classify the other classes with an acceptable accuracy (Laurikkala 2001).

4.2 Results

4.2.1 RQ1 - Purpose of Developer Instant Messages in *GitterCom*

Table 6 shows the number of messages per category across the 10 systems in our dataset and Fig. 2 shows the overall percentage distribution across categories for the entire *GitterCom* dataset. Our results show the purpose of messages based on the purpose categories defined by Lin et al. (2016) as we did not encounter any messages that required the creation of new categories to describe their purpose. However, a larger dataset which samples different communities and users may lead to different results and messages that require the creation of additional categories needed to describe their purpose.

As seen in Fig. 2, the percentages by category vary greatly. Overall, 83% of the messages are meant to support activities directly associated with the development of the system(*i.e.*, communication, Dev-Ops, team collaboration, and customer support), 14.31% of the messages are related to engagement with communities of practice, and only 2.69% of the messages are linked to personal benefits (*i.e.*, discovery and news, fun, and networking and social activities). Moreover, about 52.75% of the messages involve communication between the developers and stakeholders, 27.75% of the messages communicate updates

regarding the status of the system, and 1.53% of the messages involve customer support. Based on the hierarchy presented in Table 1, we notice that developers' use of Gitter instant messages in practice focuses mostly on team-wide purposes (83% of all messages).

We found that in the communication category, 74.2% of the messages involve communication between team members and 25.8% of the messages discuss non-work related topics. We did not find any messages related to communication with other stakeholders other than developers and customers.

Another interesting finding is that although the channels in our dataset are dedicated to the active development of a particular system, we can see some instances of messages by developers or customers of the system inquiring about particular technologies and frameworks or to bounce ideas on how to implement something using either the system in question or a related technology.

The high presence of messages that communicate updates regarding the status of the system (*i.e.*, Dev-Ops) show that developers still need to communicate some of the updates such as bug fixes despite the presence of integrations with GitHub and JIRA within Gitter. This finding could indicate that in their current state, these issue tracking tools or their integrations may not be displaying all the relevant information. However, further research is needed to assess whether this is the case and to determine the information required to provide meaningful project status update notifications to other developers.

Moreover, the low percentage of messages associated with customer support shows that despite the Gitter channels being open for users of the system to obtain support, Gitter is not a widely adopted medium for the users of these communities to directly contact the developers for this purpose. We can further find evidence of this phenomenon in Table 6, where we see that some of the channels did not have any messages related to customer support. In some communities such as *cucumber*, the lack of this type of messages in our dataset can be explained by our data collection focusing on the main channel and the presence of a channel dedicated to people asking questions. However, other communities such as ImageJ, Hibernate and FreezingMoon do not have a dedicated channel for customer support.

Additionally, as seen in Table 6, the majority of the systems have the largest category of messages dedicated to communication between developers, which is also reflected in the overall trend shown in Fig. 2. However, by focusing on individual channels, we can observe that half of the channels also have a large amount of messages dedicated to customer support (*i.e.*, Cucumber, ImageJ, JHipster, TheHolyWaffle, and UIKIT). These observations indicate that further work is needed in the analysis of these communications to determine potential barriers or facilitators for customers when using these platforms to obtain support.

We believe the reason for the low presence of messages related to personal benefits and community support purposes in our dataset is due, in part, to the nature of the channels themselves. In particular, we observed that channels for active open source systems follow certain informal rules to keep the use of these particular channels focused on team-wide purposes. For instance, the scikit-learn Gitter channel description reads, "*...Please feel free to ask specific questions about scikit-learn. Please try to keep the discussion focused on scikit-learn usage and immediately related open source projects from the Python ecosystem.*" Moreover, in some cases, some of the developers act as content moderators by asking people to keep the content of the channel relevant, "*please let's keep this chat uikit related, thanks :-)*" or, "*I'd like to keep stuff in this channel in English so everyone present understands what's being talked about.*" These informal rules and moderation limit the use of these channels for other purposes.

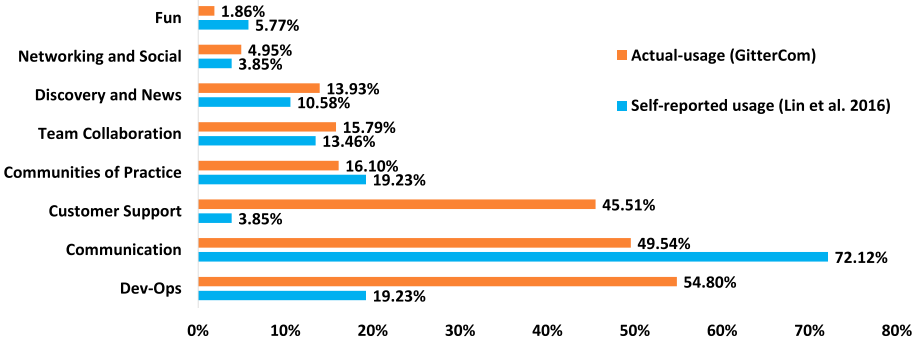

Fig. 3 Self-reported usage and actual usage per purpose

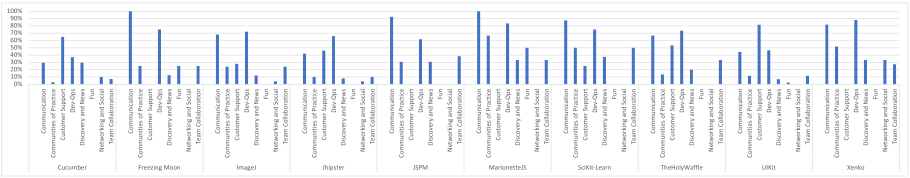

Fig. 4 Usage per purpose for each system in *GitterCom*

Our analysis indicates that the use of instant-messaging tools by software developers will vary depending on the system and type of channel they interact with. In particular, based on the content of the messages, we see that the purpose of messages posted within the channels dedicated to the development of open-source software systems is mostly focused on the development of the system itself (*i.e.*, team-wide purposes). We believe that other purposes are likely to be more prevalent in separate, dedicated channels (*e.g.*, channels that focus on communities of practice such as the ones explored by Chatterjee et al. (2019, 2020)).

4.2.2 RQ2 - Comparison Between Developer Self-Reported Usage and Actual Usage of Instant Messages

Figure 3 shows the contrast between the percentage of developers that self-reported using Slack for a particular purpose in Lin et al. (2016) and the percentage of users that actually used instant messages in *GitterCom* for the same purpose. A user is considered to use the messaging platform for a particular purpose if they authored at least one message categorized with that purpose in *GitterCom*. Note that the percentages across purposes can add to more than 100% as each user can use these instant-messaging communication systems for more than one purpose.

As seen in Fig. 3, the percentages of self-reported usage and actual-usage by purpose vary in some categories but are similar in others. In particular, we see that the distribution of usage for networking and social, discovery and news, team collaboration, and communities of practice are rather similar, whereas the distribution of usage for fun, customer support, communication, and Dev-Ops are different. More specifically, there is a considerably higher usage for purposes like Dev-Ops and customer support in *GitterCom* compared to the self-reported usage, which is to be expected in communities that actively support a

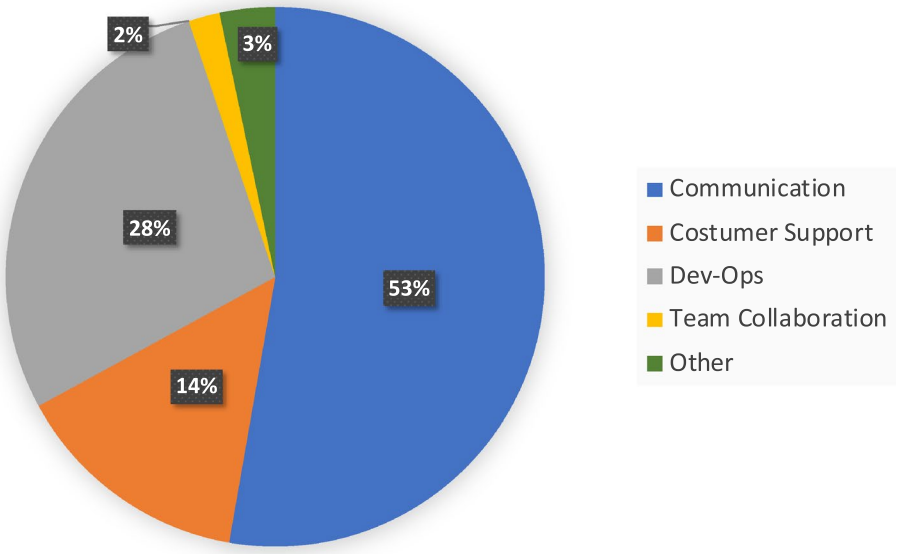

Fig. 5 Distribution of the ground truth messages into the five categories used for automatic classification

software system such as the ones in *GitterCom*. Conversely, we see a considerably lower percentage of developers in *GitterCom* using the platform for fun and general communication, specifically for communication associated with non-work topics. These differences are somewhat to be expected due to the rules and moderation generally present in the channels in *GitterCom* that aim to keep the content of the channels relevant to the system being developed, thus reducing the number of messages related to fun and communication about non-work topics.

The similar distribution of self-reported and actual usage for networking and social, discovery and news, team collaboration, and communities of practice purposes suggests that *GitterCom* is representative of developer communications via instant messaging platforms that align with these purposes and that they are likely to be present to a similar degree whenever chat-based instant messaging is used by developers.

GitterCom was derived from labeling data from actual communities rather than derived from interviewing developers. Therefore, we are able to take a closer look at the usage in *GitterCom* to provide an initial insight on how developers use chat-based communication platforms, by looking at the usage per category at a system level. For this purpose, we performed the same analysis of developers' actual usage across *GitterCom*, but aggregating usage per system. Is it to note that this represents the usage by users in our dataset and not all the users in the corresponding channels. In particular, it represents the usage by 111 users in Cucumber, 8 users of Freezing Moon - Ancient Beast, 25 users in ImageJ, 13 users in JSPM, 50 users in Jhipster, 6 users in MarionetteJS, 8 users in SciKit-Learn, 15 users in TheHolyWaffle, 43 users in UIKit, and 33 users in Xenko. Figure 4 shows the contrast between the percentage of developers that used Gitter for each purpose in each system in *GitterCom*. Although *GitterCom* is the largest dataset of its kind, it is still limited in size and the results from this analysis can only provide an initial insight into developers' instant messaging actual usage in different communities, which can be used as a starting point for further research into chat-based communications and communities.

Table 7 Performance of the machine learning classifiers on *GitterCom*. LSTM* = LSTM with SE word embeddings; CNN* = CNN with SE word embeddings

Sampling strategy	None			SMOTE			Tomek Links			NCL		
Classifier	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec	Acc	Pre	Rec
Random	0.37	0.24	0.24	0.24	0.25	0.22	0.37	0.24	0.24	0.73	0.25	0.25
SGD	0.54	0.33	0.26	0.38	0.28	0.24	0.53	0.42	0.28	0.87	0.69	0.41
Naive Bayes	0.53	0.32	0.26	**0.53**	0.28	0.27	0.53	0.32	0.25	0.86	0.60	0.47
AdaBoost	0.53	0.34	0.27	0.47	0.29	0.26	0.53	0.26	0.28	0.86	0.62	0.43
SVM	0.53	0.11	0.20	0.48	0.28	0.26	0.53	0.13	0.25	0.84	0.21	0.25
Random Forest	0.53	0.38	0.29	0.43	0.28	0.25	0.53	0.41	0.29	**0.88**	0.69	0.56
Decision Trees	0.52	0.38	0.30	0.45	0.29	0.25	0.53	0.40	0.29	**0.88**	0.70	0.57
K-NN	0.43	0.27	0.25	0.47	0.30	0.26	0.51	0.44	0.28	0.80	0.63	0.46
LSTM	**0.58**	0.50	0.40	0.41	0.51	0.47	0.56	0.46	0.40	0.75	0.47	0.31
CNN	0.53	0.38	0.30	0.41	0.28	0.30	**0.58**	0.40	0.47	0.74	0.39	0.34
LSTM*	0.53	0.15	0.23	**0.53**	0.12	0.22	0.53	0.12	0.22	0.53	0.12	0.22
CNN*	0.53	0.21	0.24	0.32	0.21	0.28	0.52	0.20	0.24	0.53	0.18	0.24

Bolded entries indicate the highest accuracy given each sampling strategy

Upon inspecting the usage distribution per system we can see that the majority of the systems in *GitterCom* follow a similar distribution to the one seen in Fig. 4. In particular, Communication, Dev-Ops, and Team Collaboration are the categories with the highest usage, whereas developers rarely use these communication channels for Fun, Discovery of news, and Networking purposes. Nonetheless, there are communities that do not follow the general distribution. For example, the UIKIT community shows a higher participation in customer support than Dev-Ops.

Overall, our results show that there are differences in how developers perceive and self-report using instant messaging platforms (based on the study by Lin et al. (2016)) and how they actually use them in *GitterCom*. The results show that across different communities, the most frequent usages of chat-based platforms are general purpose communication and updates regarding the software system (*i.e.*, Dev-Ops). This phenomenon can be a bi-product of the more focused goal and nature of the OSS development communities in *GitterCom*.

4.2.3 RQ3 - Automatic Classification of Developer Instant Messages

The distribution of the ground truth data in the 5 categories used in the automatic classification process is shown in Fig. 5.

Columns 2-4 of Table 7 show the results of the classification algorithms in terms of accuracy, precision, and recall. We can see that all the classifiers outperform the random classifier used as the baseline. In particular, the random classifier achieves an accuracy of 0.37, whereas, all the other classifiers achieve accuracies ranging between 0.43 and 0.58. The top classifiers are LSTM, with an accuracy of 0.58, followed by Stochastic Gradient Descent which achieved an accuracy of 0.54. However, despite outperforming the random baseline, the overall performance of the classifiers is still low which would limit the applicability of the automatic classification on Gitter channels.

After inspecting more closely the results of the classifiers, we identified that their performance was likely hindered by misclassifications caused by 1) vocabulary overlap between the majority class and the second-largest class and 2) the imbalanced distribution of the training data among the classes (see Fig. 5), causing all of the classifiers to be biased towards the majority class.

To address the negative effects of data imbalances and non-linear separability in our dataset, we performed experiments with SMOTE as oversampling algorithm, and NCL and the removal of Tomek links as under-sampling algorithms. Columns 5-13 of Table 7 show the results for each of these experiments. Regarding the performance of the classifiers, our results show that over-sampling the smaller classes negatively impacts the performance of the classifiers, the removal of Tomek links does not have a major impact on the performance, and under-sampling using NCL improves the performance of all the classifiers.

SMOTE has a negative impact on the classifiers because although it addressed the imbalance of the dataset, it generated synthetic data that fell between the two largest classes, therefore decreasing their linear separability.

As shown in Table 7, using the NCL under-sampling algorithm achieves higher performance than using over-sampling or the removal of Tomek Links under-sampling. It is to note that the neural networks classifiers only slightly outperform the baseline classifier in terms of accuracy after applying the NCL under-sampling algorithm. This result could be explained by neural network classifiers' performance being reliant on the availability of large amounts of data for training (Lai et al. 2015), while NCL reduces the amount of training data available.

The overall higher performance of the NCL sampling strategy is due to the NCL algorithm being designed to prevent the classifiers from overfitting towards the majority classes (Laurikkala 2001), whereas the synthetic data produced by SMOTE solves the data imbalance issue but it does not address the linear separability of the samples. We see that the best performing classifiers after applying the NCL under-sampling algorithm are Random Forest and Decision Trees.

When using the software engineering word embeddings in the deep learning approaches (LSTM* and CNN* in Table 7) we found that the neural networks achieved a lower performance across all the sampling approaches. The reduced performance may be due to the specificity of the word embeddings which seems to hinder the ability of the neural networks to classify the messages which contain a large amount of non-software engineering specific tokens. Future research in the field could benefit from exploring a combination of the software engineering word embeddings with dataset-specific word embeddings.

We further investigate our results by applying statistical tests on the performance of the approaches. In particular, we first determined if the distributions of the classifiers' accuracy were normally distributed using the Shapiro-Wilk test with a significance level of 0.01 and found that they were not normally distributed. Next, we compare the accuracies of the best performing classifiers (Random Forest and Decision Trees) with those of each of the other machine learning classifiers, using the one-sample Mann-Whitney U test (since the observations were not normally distributed) and apply the Bonferroni correction to adjust the p-values. The results show that the difference between the performance of Random Forest and all the other classifiers is statistically significant, except for Decision Trees, as confirmed by the results of the Mann-Whitney U tests with 95% confidence interval (p-values < 0.05 and a large Cliff's delta

effect size). The tests indicate no statistically significant difference between Decision Trees and Random Forest.

5 Potential Research Applications

Previous studies have investigated the growing use of alternative communication means by developers (Lin et al. 2016; Käfer et al. 2018; Stray et al. 2019). The results of these studies show the rise of instant messaging tools and the impact they have on reshaping team dynamics and the communication landscape in increasingly distributed software development environments. Future studies could make use of *GitterCom* to study the relationship between open source development activity and communication trends in chat based platforms. In particular, *GitterCom* enables further research to analyze and understand patterns in developer communications such as the works by Ehsan et al. (2020); Sahar et al. (2020) and to address important questions such as: How do software teams use tools like Gitter to communicate among themselves and with other stakeholders? How do team dynamics reflect in team communications? Do developers exchange different types of messages at different times in the software life cycle? Do developers new to a project post different types of messages than the more senior developers? How can we facilitate users of the systems to communicate directly with the developers through these communication platforms?

GitterCom can also be used as a training dataset for machine learning approaches for automatic classification of new messages based on their purpose. The study presented in Section 4 lays the foundation for this and shows that such a classification is feasible, with a relatively high accuracy.

A potential application of *GitterCom* would be to leverage the data to automatically organize messages into threads or to create summaries of developer conversations based on their purpose, such that developers that were away for a while or newcomers to a project could quickly catch up on important conversations they missed. Machine learning applications could further complement the developers' workflow with information from external sources related to the messages exchanged, such as Stack Overflow.

Another avenue for future work would be to use *GitterCom* in order to perform large scale replications of previous studies that analyzed developer communications in Slack (Alkadhi et al. 2017a; Chatterjee et al. 2019), but used much smaller or restricted datasets (*e.g.*, communications in student projects or a particular software company). These replications on *GitterCom* could help corroborate previous findings or uncover new information about how developers communicate through instant messaging tools. One example of such work that could benefit from a large scale replication is work on the identification of messages that contain rationale for the decisions made by developers throughout the software life cycle (Alkadhi et al. 2017b). Thus far, work on rationale has been limited to analyzing the chat messages of three student teams working on a multi-project capstone course.

6 Threats to Validity

In this section, we describe the threats to validity we faced in our work and how we addressed them.

Regarding threats to external validity, due to the size of the dataset, our results may not be generalizable to all the software development chat rooms available.It is to note that in this work, we focused on Gitter channels associated with open source software systems and we only manually analyzed a subset of all the messages within the selected channels and communities. Therefore, the messages in our dataset may not be representative of all developers or of all messages exchanged by developers in all the software development instant messaging platforms and communities. Our study also focuses on Gitter channels that are linked to GitHub repositories as Gitter did not have the ability to directly link to other hosting services at the time of data collection, therefore the results may not be reflective of newer Gitter channels that are linked to GitLab repositories. Moreover, the use of modern enterprise chat room tools is likely to vary in channels that focus on communities of practice, such as the ones explored by Chatterjee et al. (2019). Nonetheless, we aimed to increase the generalizability of our findings by including channels of various sizes that cover a wide variety of application domains. We used a set of selection criteria for the systems included in *GitterCom* to ensure that the chosen channels and the results are focused on communications surrounding open source software development. However, due to our focus on the main channels within these communities, which showed a higher user activity, our results might not be reflective of other channels with a lower level of activity within the same communities. Furthermore, it is possible that the distribution of messages across different purposes in *GitterCom* is not representative of all developer communications. We mitigated this threat by considering a varied set of communities in the dataset.

Although our results did not require the creation of new categories to describe the messages in *GitterCom*, future work may require new categories to be created in addition to those derived by Lin et al. (2016) as the communication of developers evolves over time.

Moreover, our approach can be applied to any existing Gitter channel. The imbalance of our dataset is another threat to validity. To mitigate the impact of class imbalance, we applied robust over-sampling approaches as well as under-sampling approaches, since it has been shown that over-sampling can cause over-fitting for classical machine learning models, especially for minority classes (Buda et al. 2018). Moreover, since under-sampling decreases the amount of data available for training it may lead to loss of information about the majority class. To mitigate this risk, we applied the NCL under-sampling algorithm which aims to prevent the classifiers from over-fitting towards the majority classes while minimizing any impact on the classification performance of the majority classes (Laurikkala 2001).

Threats to internal validity we faced in this work relate to whether there is sufficient evidence to support our findings. The main threat to validity is that the categorization of the messages in *GitterCom* does not properly reflect the purpose of the developer messages. Moreover, when comparing the purpose of messages within a community, the number of users in our dataset represents only a part of all the users in the communities. Therefore, future work analyzing a larger sample of users or different communities may lead to different results. In order to mitigate this threat, we leveraged a set of categories derived from previous work as a coding guide with detailed instructions, and had two raters independently label each message. This was followed by the raters meeting to resolve any conflicts and update the coding guide accordingly, resulting in an inter-rater agreement of 0.89, which is considered high. The second aspect that could negatively impact the internal validity of the results is the sampling strategy when selecting the messages to label. We mitigated this threat by selecting the most recent messages rather than random selection in order to ensure the data reflects current practices and that *GitterCom* retains the historical/sequential information associated with chat communications.

Threats to construct validity refer to how we measured the results of our studies. We mitigated threats to construct validity in each study by:

(i) Using an established set of categories from previous work and having two authors individually annotate the dataset and then discussing any discrepancies that occurred,
(ii) Using well-tested implementations of the machine learning classifiers and sampling algorithms,
(iii) Employing common performance evaluation metrics, and
(iv) Using a robust and established evaluation framework from the Natural Language Processing field with 10-fold cross-validation and hyper-parameter tuning (Liu and Liu 2008).

7 Conclusions and Future Work

We introduced a dataset, called *GitterCom*, which is the largest manually labeled and curated dataset of software development communications through instant messaging platforms to date. *GitterCom* contains 10,000 Gitter messages collected from the main Gitter channels associated with the development of 10 open source software systems. The messages were manually labeled according to their purpose, using the message classification hierarchy first introduced by Lin et al. (2016).

Our results indicate that developers and other stakeholders use the Gitter channels dedicated to open source systems mostly for activities related to the active development of the system. Our findings indicated that Gitter is also used by developers as an alternative to communication platforms like Stack Overflow or forums to inquire about new technologies or coding challenges they face. Lastly, only a small percentage of the use of these channels was devoted to personal benefits or community support. We also found that the distribution of messages across various purposes in the channels dedicated to open source development can vary significantly from system to system. Furthermore, when comparing the actual-usage in *GitterCom* with the self-reported usage by developers based on the work of Lin et al. (2016), we found that they present some similarities and differences. The differences mostly consist of a higher percentage of messages related to Dev-Ops and Customer Support, as well as a lower percentage of messages dedicated to participation in communities of practice, fun, and communication about non-work topics in *GitterCom* compared to Lin et al. (2016). These differences could be explained by the fact that the channels included in *GitterCom* are dedicated to open source development, whereas the results in Lin et al. (2016) are based upon responses to a more general question on the use of Slack by developers. Moreover, by analyzing the usage on a per-channel basis, we found that the majority of channels in our dataset follow a similar distribution to that of *GitterCom* as a whole.

In their analysis of the use of a single Slack community within a large company, Stray et al. (2019) indicated that the team members should agree upon and communicate to new members clearly defined guidelines on how to use the instant messaging tools. During our study of *GitterCom*, we found that some open source development teams using Gitter as their instant messaging tool have such guidelines in place and that the members of the development teams actively work to ensure these guidelines are followed. However, we also noticed that these guidelines are focused on keeping the information exchanged in the

communities related to the development of the system and not on how to organize the community itself.

We found that the nature and frequency of the messages exchanged in software development Gitter chat rooms have an intrinsic imbalance and language overlap across purposes, making it difficult for machine learning classifiers to perform well without dealing with the class imbalance problem. We found that machine learning classifiers using under-sampling achieve higher performance and up to 88% accuracy, which is a very promising result for this multi-class classification problem.

Our future work will focus on improving the performance of the classifiers by using a larger amount of data. Moreover, future avenues of research include (i) studying how the structure of the information exchanged through these channels may impact developers' engagement with other team members and users of the systems, (ii) the design and implementation of effective mechanisms to present relevant information from instant messaging tools that align with the developers' purpose.

Another avenue for future work would be to develop mechanisms to allow the developers and other users to manually indicate the purpose of a message upon its creation, similar to the way Hyperdialog (Gomes Pimentel et al. 2003) and REACT (Alkadhi et al. 2017a) record the conversation threads and rationale of messages, respectively.

References

Alkadhi R, Johanssen JO, Guzman E, Bruegge B (2017a) REACT: An Approach for Capturing Rationale in Chat Messages. In: Proceedings of the 11th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM'17), pp 175–180

Alkadhi R, Lata T, Guzmany E, Bruegge B (2017b) Rationale in development chat messages: An exploratory study. In: Proceedings of the 14th IEEE/ACM international conference on mining software repositories (MSR'17), pp 436–446

Allamanis M, Sutton C (2013) Why, when, and what: Analyzing stack overflow questions by topic, type, and code. In: Proceedings of the 10th IEEE working conference on mining software repositories (MSR'13), pp 53–56

Aly M (2005) Survey on multiclass classification methods. Neural Network 19:1–9

Anders A (2016) Team communication platforms and emergent social collaboration practices. Int J Business Commun 53(2):224–261

Arora P, Ganguly D, Jones GJF (2015) The good, the bad and their Kins: Identifying questions with negative scores in StackOverflow. In: Proceedings of the 2nd IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM'15), pp 1232–1239

Bergstra J, Bengio Y (2012) Random Search for Hyper-Parameter Optimization. J Mach Learn Res 13(2):281–305

Beyer S, Macho C, Pinzger M, Di Penta M (2018) Automatically classifying posts into question categories on stack overflow. In: Proceedings of the 26th IEEE international conference on program comprehension (ICPC'18), Association for Computing Machinery, New York, NY, USA, ICPC '18, pp 211–221 https://doi.org/10.1145/3196321.3196333

Bottou L (2010) Large-Scale Machine Learning with Stochastic Gradient Descent. In: Lechevallier Y, Saporta G (eds) Proceedings of the 19th international conference on computational statistics (COMPSTAT'10), pp 177–186

Breiman L (2001) Random Forests. Mach Learn 45(1):5–32

Buda M, Maki A, Mazurowski MA (2018) A Systematic Study of the Class Imbalance Problem in Convolutional Neural Networks. Neural Networks 106:249–259

Chatterjee P, Damevski K, Pollock L, Augustine V, Kraft NA (2019) Exploratory study of slack Q&A chats as a mining source for software engineering tools. In: Proceedings of the 16th IEEE international conference on mining software repositories (MSR'19), pp 490–501

Chatterjee P, Damevski K, Kraft NA, Pollock L (2020) Software-related slack chats with disentangled conversations. In: Proceedings of the 17th IEEE international conference on mining software repositories (MSR'20), pp 588–592

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: Synthetic Minority Over-sampling Technique. J Artif Intell Res 16(1):321–357

Chowdhury SA, Hindle A (2015) Mining StackOverflow to filter out Off-topic IRC discussion. In: Proceedings of the 12th IEEE working conference on mining software repositories (MSR'15), pp 422–425

Constantino K, Zhou S, Souza M, Figueiredo E, Kastner C (2020) Understanding collaborative software development: An interview study. In: Proceedings of the 15th ACM/IEEE international conference on global software engineering (ICGSE'20), pp 55–65

Cortes C, Vapnik V (1995) Support-vector Networks. Mach Learn 20(3):273–297

cuc (2020) Cucumber. https://github.com/cucumber/cucumber

Deng L (2014) A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning. Transactions on Signal and Information Processing 3

Efstathiou V, Chatzilenas C, Spinellis D (2018) Word embeddings for the software engineering domain. In: Proceedings of the 15th IEEE international conference on mining software repositories (MSR'18), MSR '18, p 38–41 DOI: https://doi.org/10.1145/3196398.3196448

Ehsan O, Hassan S, Mezouar ME, Zou Y (2020) An Empirical Study of Developer Discussions in the Gitter Platform. TOSEM pp 1–39

Elsner M, Charniak E (2011) Disentangling chat with local coherence models. In: Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies (ACL'11), pp 1179–1189

Fang H, Klug D, Lamba H, Herbsleb J, Vasilescu B (2020) Need for tweet: How open source developers Talk about their GitHub work on twitter. In: Proceedings of the 17th IEEE international conference on mining software repositories (MSR'20), pp 322–326

Ford D, Lustig K, Banks J, Parnin C (2018) "We Don't Do That Here": How collaborative editing with mentors improves engagement in social Q&A communities. In: Proceedings of the 2018 conference on human factors in computing systems (CHI'18)', pp 1–12

fre (2020) Freezingmoon. https://github.com/FreezingMoon

Gers FA, Schraudolph NN, Schmidhuber J (2002) Learning Precise Timing with LSTM Recurrent Networks. J Mach Learn Res 3(1):115–143

Gomes Pimentel M, Fuks H, de Lucena CJP (2003) Co-text loss in textual chat tools. In: Procceedings of the 4th international and interdisciplinary conference on modeling and using context (CONTEXT'03), pp 483–490

Guzman E, Ibrahim M, Glinz M (2017) A little bird told me: Mining tweets for requirements and software evolution. In: Proceedings of the 25th IEEE international requirements engineering conference (RE'17), pp 11–20

Hastie T, Rosset S, Zhu J, Zou H (2009) Multi-class AdaBoost. Statistics and Its. Interface 2(3):349–360

ima (2020) Imagej. https://github.com/imagej/imagej

jhi (2020) jhipster. https://github.com/jhipster/jhipster/

jsp (2020) jspm. https://github.com/jspm

Käfer V, Graziotin D, Bogicevic I, Wagner S, Ramadani J (2018) Communication in Open-Source Projects-End of the E-mail Era? In: Proceedings of the 40th IEEE/ACM international conference on software engineering(ICSE'18), pp 242–243

Keivanloo I, Rilling J, Zou Y (2014) Spotting working code examples. In: Proceedings of the 36th IEEE international conference on software engineering (ICSE'14), pp 664–675

Khan A, Baharudin B, Lee LH, Khan K (2010) A Review of Machine Learning Algorithms for Text-Documents Classification. J Adv Inform Technol 1(1):4–20

Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the 11th SIGDAT conference on empirical methods in natural language processing (EMNLP'14), pp 1746–1751

Lai S, Xu L, Liu K, Zhao J (2015) Recurrent convolutional neural networks for text classification. In: Proceedings of the 29th AAAI conference for artificial intelligence (AAAI'15), pp 2267–2273

Laurikkala J (2001) Improving identification of difficult small classes by balancing class distribution. In: Conference in artificial intelligence in medicine in Europe (AIME'01), Lecture Notes in Computer Science, pp 63–66

Lin B, Zagalsky A, Storey MA, Serebrenik A (2016) Why developers are slacking off: Understanding how software teams use slack. In: Proceedings of the 19th ACM conference on computer supported cooperative work and social computing (CSCW'16), pp 333–336

Linares-Vasquez M, Dit B, Poshyvanyk D (2013) An exploratory analysis of mobile development issues using stack overflow. In: Proceedings of the 10th IEEE working conference on mining software repositories (MSR'13), pp 93–96

Liu F, Liu Y (2008) Correlation between ROUGE and human evaluation of extractive meeting summaries. In: Proceedings of the 46th ACL annual meeting of the association for computational linguistics on human language technologies (HTL'08), pp 201–204

mar (2020) Marionette. https://github.com/marionettejs/backbone.marionette

McCallum A, Nigam K (1998) A comparison of event models for naïve bayes text classification. In: Proceedings of the 1st AAAI workshop on learning for text categorization (ICML/AAAI'98), pp 41–48

Mikolov T, Deoras A, Povey D, Burget L, Cernocký J (2011) Strategies for training large scale neural network language models. In: Proceedings of the 12th IEEE workshop on automatic speech recognition understanding (ASRU'11), pp 196–201

Murgia A, Janssens D, Demeyer S, Vasilescu B (2016) Among the machines: Human-Bot interaction on social Q&A websites. In: Proceedings of the 2016 conference extended abstracts on human factors in computing systems (CHI/EA'16), pp 1272–1279

Nasehi SM, Sillito J, Maurer F, Burns C (2012) What makes a good code example?: A study of programming Q&A in StackOverflow. In: Proceedings of the 28th IEEE international conference on software maintenance (ICSM'12), pp 25–34

Novielli N, Calefato F, Lanubile F (2014) Towards discovering the role of emotions in stack overflow. In: Proceedings of the 6th international workshop on social software engineering, SSE'2014, pp 33–36

Novielli N, Calefato F, Lanubile F (2015) The challenges of sentiment detection in the social programmer ecosystem. In: Proceedings of the 7th international workshop on social software engineering (SSE'15), pp 33–40

Paikari E, van der Hoek A (2018) A framework for understanding chatbots and their future. In: Proceedings of the 11th international workshop on cooperative and human aspects of software engineering (CHASE'18), pp 13–16

Panichella S, Di Sorbo A, Guzman E, Visaggio C, Canfora G, Gall H (2015) How can I improve my App? Classifying user reviews for software maintenance and evolution. In: Proceedings of the 31st IEEE international conference on software maintenance and evolution (ICSME'15), pp 281–290

Parra E (2020) Gittercom, dataset. https://doi.org/10.6084/m9.figshare.11626008

Parra E (2021) Replication package. https://figshare.com/s/576d328da4a5b50ea155

Parra E, Ellis A, Haiduc S (2020) GitterCom - A dataset of open source developer communications in gitter. In: Proceedings of the 17th IEEE international conference on mining software repositories (MSR'20), pp 563–567

Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: Empirical methods in natural language processing (EMNLP), pp 1532–1543

Poché EH (2017) Analyzing User Comments On YouTube Coding Tutorial Videos. mathesis, Louisiana State University, Baton Rouge, LA, USA

Ponzanelli L, Mocci A, Bacchelli A, Lanza M (2014) Understanding and classifying the quality of technical forum questions. In: Proceedings of the 14th international conference on quality software (QSIC'14), pp 343–352

Rekha S, Divya N, Bagavathi S (2014) A hybrid auto-tagging system for StackOverflow forum questions. In: Proceedings of the 1st international conference on interdisciplinary advances in applied computing (ICONIAAC'14), pp 1–5

Russell SJ, Norvig P (1995) Artificial Intelligence: A Modern Approach

Safavian SR, Landgrebe D (1991) A survey of Decision Tree Classifier Methodology. IEEE Trans Syst Man Cybern 21(3):660–674

Sahar H, Hindle A, Bezemer CP (2020) How are Issue Reports Discussed in Gitter Chat Rooms? Journal of Systems and Software pp 110852, https://doi.org/10.1016/j.jss.2020.110852., http://www.sciencedirect.com/science/article/pii/S0164121220302429

Sajedi Badashian A, Hindle A, Stroulia E (2016) Crowdsourced bug triaging: Leveraging Q&A platforms for bug assignment. In: fundamental approaches to software engineering, lecture notes in computer science, pp 231–248

sci (2020) scikit-learn. https://github.com/scikit-learn/scikit-learn

Seiffert C, Khoshgoftaar TM, Van Hulse J, Folleco A (2014) An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data. Inform Sci 259(1):571–595

Shi L, Chen X, Yang Y, Jiang H, Jiang Z, Niu N, Wang Q (2021) A first look at developers' live chat on gitter. In: Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering (ESEC/FSE'21), Association for Computing Machinery, pp 391–403 https://doi.org/10.1145/3468264.3468562

Shihab E, Jiang ZM, Hassan AE (2009) Studying the use of developer IRC meetings in open source projects. In: Proceedings of the IEEE international conference on software maintenance (ICSM'09), pp 147–156

Singh A, Thakur N, Sharma A (2016) A review of supervised machine learning algorithms. In: Proceedings of the 3rd IEEE international conference on computing for sustainable global development (INDIACom'16), pp 1310–1315

Storey M, Zagalsky A, Filho FF, Singer L, German DM (2017) How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. IEEE Trans Softw Eng 43(2):185–204

Storey MA, Zagalsky A (2016) Disrupting developer productivity one bot at a time. In: Proceedings of the 24th ACM/SIGSOFT international symposium on foundations of software engineering (FSE'16), pp 928–931

Storey MA, Singer L, Cleary B, Figueira Filho F, Zagalsky A (2014) The (R) Evolution of social media in software engineering. In: Proceedings of the 36th ACM/IEEE international conference in software engineering, future of software engineering (FOSE'14), pp 100–116

Stray V, Moe NB, Noroozi M (2019) Slack me if you can!: Using enterprise social networking tools in virtual agile teams. In: Proceedings of the 14th IEEE international conference on global software engineering (ICGSE'19), pp 101–111

Subramanian S, Holmes R (2013) Making sense of online code snippets. In: Proceedings of the 10th IEEE working conference on mining software repositories (MSR'13), pp 85–88

thw (2020) Theholywaffle. https://github.com/TheHolyWaffle

Tian Y, Lo D, Lawall J (2014) Automated construction of a Software-Specific word similarity database. In: Proceedings of the 1st joint meeting of The IEEE conference on software maintenance, reengineering, and reverse engineering (CSMR-WCRE'04), pp 44–53

Tomek I (1976) Two Modifications of CNN. IEEE Transactions on Systems, Man, and Cybernetics SMC–6(11):769–772

Treude C, Barzilay O, Storey MA (2011) How do programmers ask and answer questions on the Web? In: Proceedings of the 33rd IEEE/ACM international conference on software engineering (ICSE'11), pp 804–807

uik (2020) uikit. https://github.com/uikit/uikit

Vassallo C, Panichella S, Di Penta M, Canfora G (2014) CODES: Mining source code descriptions from developers discussions. In: Proceedings of the 22nd IEEE international conference on program comprehension (ICPC'14), pp 106–109

West D (2000) Neural Network Credit Scoring Models. Computers & Operations Research 27(11):1131–1152

xen (2020) Xenko3d. https://gitter.im/xenko3d/xenko

Xu B, Xing Z, Xia X, Lo D (2017) Answerbot: Automated generation of answer summary to developers' technical questions. In: Proceedings of the 32nd IEEE/ACM international conference on automated software engineering (ASE'17), pp 706–716