Multimodal Recommendation of Messenger Channels

Ekaterina Koshchenko
ekaterina.koshchenko@jetbrains.com
JetBrains Research
Amsterdam, The Netherlands

Egor Klimov
egor.klimov@jetbrains.com
JetBrains Research
Saint Petersburg, Russia

Vladimir Kovalenko
vladimir.kovalenko@jetbrains.com
JetBrains Research
Amsterdam, The Netherlands

ABSTRACT

Collaboration platforms, such as GitHub and Slack, are a vital instrument in the day-to-day routine of software engineering teams. The data stored in these platforms has a significant value for data-driven methods that assist with decision-making and help improve software quality. However, the distribution of this data across different platforms leads to the fact that combining it is a very time-consuming process. Most existing algorithms for socio-technical assistance, such as recommendation systems, are based only on data directly related to the purpose of the algorithms, often originating from a single system.

In this work, we explore the capabilities of a multimodal recommendation system in the context of software engineering. Using records of interaction between employees in a software company in messenger channels and repositories, as well as the organizational structure, we build several channel recommendation models for a software engineering collaboration platform, and compare them on historical data. In addition, we implement a channel recommendation bot and assess the quality of recommendations from the best models with a user study.

We find that the multimodal recommender yields better recommendations than unimodal baselines, allows to mitigate the overfitting problem, and helps to deal with cold start. Our findings suggest that the multimodal approach is promising for other recommendation problems in software engineering.

ACM Reference Format:
Ekaterina Koshchenko, Egor Klimov, and Vladimir Kovalenko. 2022. Multimodal Recommendation of Messenger Channels. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524842.3528441

1 INTRODUCTION

These days, software professionals working in organizations use various collaboration platforms almost daily (messengers, version control system (VCS) hosting platforms, task trackers, etc.) [1]. These platforms record and keep a lot of information describing how users interact with each other and various platform entities. Combining such data can provide a good representation of social and technical connections within organizations [2]. These representations are used in techniques and tools that support collaborative work in teams, such as automated bug assignment tools [3] or reviewer recommendation algorithms [4]. We collectively refer to such techniques as *socio-technical assistance techniques*.

Expert recommendation systems [5], such as code reviewer recommendation [4], are an example of socio-technical assistance algorithms. Reviewers have a direct impact on the quality of the code, as well as the programmers' growth [6]. Moreover, proper selection of reviewers is vital for maintaining the pace of development [4] Another application of socio-technical assistance algorithms is maintaining a healthy bus factor [7, 8] by keeping individual contributors from becoming indispensable [9].

Another application for socio-technical assistance algorithms recommendation of channels in text messengers. A *channel* is a group chat where all communication is focused on specific projects, topics, or teams. Slack [10] is a popular corporate messenger featuring the channels model. Channels help to focus communication on specific topics in one place [11]. Although the channels model is quite convenient [12], there is a significant flaw: users are limited by the channels they already know about. Thus, it is hard for the users to look outside their "neighbouring" scope. Automatic channel recommendation systems [13] can address this issue. Relevant and diverse recommendations increase employees' awareness of each other's projects, and stimulate interactions between teams.

Intuitively, the more data about employees and their communication we use in those algorithms, the better their performance. However, for a long time different activities were scattered across many independent tools. Here is an example of such arrangement in a hypothetical team: CI builds are run in Jenkins[1], documentation is stored in Confluence, code is reviewed in Upsource, issues are tracked in Jira, developers communicate through Slack.

Such arrangement leads to the information being scattered across different platforms. This makes it much harder to collect, link, and process the data to use it to enhance the tools and processes.

Taking into account the efforts to collect and link individual data types, an increase in the number of individual tools leads to a quadratic growth of data collection complexity. It comes as no surprise that most research on socio-technical assistance algorithms focuses on one type of data [14]. For example, Slack's description of their channel recommendation system [13] suggests that they only use information about the time users spend in channels (reading and writing). One of the possible improvements over this would be to use organization's structure (hierarchy). Employees with the same role and similar projects may be interested in the same channels.

Modern developer tools combine functionalities of multiple tools into one platform. Examples of this trend include tools like Github Enterprise or Space [15]. These tools combine the functionality of several other tools, such as VCS hosting, code review tool, task

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9303-4/22/05…$15.00
https://doi.org/10.1145/3524842.3528441

[1]This and other names in this list are mainstream collaboration tools in the software industry.

tracker, calendar, messenger, or team directory, in a single tool. In these combined tools, various sorts of records of collaboration within teams are available in the same environment. This in turn makes it easier to process this data jointly, and enables *multimodal* approaches to socio-technical support systems.

In this work, we focus on multimodal recommendation of channels for messenger workspaces. We build a multimodal recommendation system based on combining three types of data: channels activity, VCS history, and organizational structure, sourced from the internal systems of JetBrains, a vendor of software engineering tools. We evaluate the system on historical data, build a Space bot around it, and compare it against best unimodal baseline approaches based on user feedback. The two primary contributions of this work are:

- We describe a channel recommendation algorithm working with three types of data: channels (their subscribers and messages grouped by threads), technical repositories (authors, commits), and organization's structure (roles, teams and management).
- We demonstrate that the multimodal recommender system provides recommendations of higher quality.

We conclude that multimodal data is of great value for the channel recommendation problem, and argue that it might also be beneficial for other recommendation tasks in software engineering and for other socio-technical assistance techniques. We share the implementation of the models and the evaluation code in the reproduction package[2].

2 BACKGROUND AND MOTIVATION

2.1 Recommender Systems in Software Engineering

Software engineering involves rapid decision-making. The engineers and their colleagues make diverse decisions at many points in their everyday activities. Examples include choosing the debugging strategy for a new issue, selecting the proper library method to call, or selecting the right colleague to ask for help or reassign a task to. While often straightforward and instant, some of these decisions are in fact of great importance: for example, selecting the right reviewers for new changesets is vital for maintaining the pace of development [4]. The research community came up with diverse techniques and tools to assist the engineering teams with decision-making in their everyday work, and, more generally, to provide them with the information relevant in their current context. Collectively, these techniques and tools are referred to as *Recommender Systems in Software Engineering (RSSE)* [16, 17].

Examples of RSSE include code recommendation and code completion systems [18, 19], expert recommendation engines, e.g. for bug assignees [3, 20] or code reviewers [4], and systems for recommendation of relevant external resources, such as libraries [21] or discussions [22].

Recommendation algorithms rely on the variety of types of data available in the engineering tools. Recommendations can be produced based on the history of technical contributions, such as code edits [4, 20], and the non-technical interactions, such as issue reassignment sequences [23] or discussions in natural language [24].

Depending on the context and the task, quality of recommendations can be evaluated by comparing their output with historical user actions [4], by observing the impact of the recommenders on users' behavior [25], or by modeling the consequences of a potential user selecting the recommended option [7].

2.2 Combining multiple data sources

Some RSSE techniques, as well as recommender systems in other domains, rely on data in various forms, often originating from different systems, or *multimodal* data. For example, a developer recommendation system by Xia et al. [26] utilizes both the textual features of the bug report and the records of developers' participation; Sulun et al. [27] propose a method for reviewer recommendation based on traceability graphs, which are built from connections between artifacts of multiple types, stored in different systems. In the context of RSSE, multimodal data is valuable because the developers and other software professionals normally use more than one tool, which means that all aspects of their preferences generally cannot be covered by, and retrieved from, the data in a single system.

Multimodal recommenders are, however, harder to build and maintain in practical scenarios. Combining data from multiple systems to feed into a multimodal recommender implies implementing and maintaining multiple data retrieval modules. Moreover, linking the data from multiple systems can be a challenge in itself [28].

At the same time, modern software engineering platforms, such as Github[1], Gitlab, Azure DevOps, and Space, combine the functionality of multiple tools — VCS hosting, issue tracker, agile boards, CI engine, code review system — in a single tool. This means that in such systems the data of different modalities is available in the same environment, which eliminates the challenge with combining the data from multiple systems. The emergence of this "swiss knife" tool model suggests that multimodal approaches to recommender systems and to other data-driven techniques for software engineering tools are now more feasible in practice than before.

2.3 Channel recommendations

The focus of this work is multimodal recommendation of messenger channels. We choose the problem of channel recommendation to tackle with a multimodal approach for two primary reasons. First, the structure of channels in companies' internal messengers is more often defined by convention rather that by a formalized process. This means that the history of messages in these channels is harder to link to structured data, such as VCS history, which in turn suggests that efficient recommendation of channels is likely to require combining more than one data source. Second, corporate messengers have only been widespread in engineering teams relatively recently. While chat applications have been the context for several existing studies [29–32], to our knowledge no scholarly research exists on approaches to recommendation of channels, with no studies to our knowledge focusing on this problem in the context of software engineering. We seek to make a dent in this literature gap with this study.

2.3.1 *Channel recommendation systems.*

A channel recommendation system in text messengers is a model that, based on information available about the user, produces suggestions for channels they might be interested in. In larger organizations, users of collaboration platforms cannot always be aware of all potentially interesting messenger channels. Relevant and diverse automatic recommendation of channels can aid software engineering professionals by:

- Reducing the time that new users spend to find channels that are related to their work and interests;
- Stimulating cross-team interactions;
- Helping employees to get more aware each other's interests and expertise;
- Raising awareness of each other's projects, which in turn can facilitate knowledge sharing.

Similarly to how traditional recommender systems leverage implicit information about the users' preferences [33], the primary source of data for channel recommendation systems is records of users' membership and communication in channels. In this study, we use two additional data modalities: version control repositories and organizational structure. VCS repositories contain technical artifacts of engineers' work. Version history also contains traces of technical collaboration of engineers, e.g. when two people work on the same feature. Organizational structure represents the roles and the hierarchy in the company. It includes teams and team memberships, roles and management connections in the company. In combination with the channel activity data, this information can help identify role-related patterns, such as "the chat for UX designers to help each other out and share memes".

Including the data on VCS history and organizational structure into channel recommendation systems might help the models make more relevant and diverse recommendations. Moreover, it might help with the cold start problem [34]. New users barely have any activity in existing channels, which means that it is not possible to generate recommendations for them based on their activity. However, new users definitely have a position in the company (defined by, for example, their team and role), and they might already have some repositories with source code. Such additional data, already reflected in internal systems, can help the recommendation models deal with the cold start problem, providing new users with relevant recommendations.

2.3.2 Channel recommendations in Slack. The most recent available description of a channel recommendation algorithm is the one published by Slack in 2016 [13]. The algorithm is a variation of item-based collaborative filtering [35]. Its overview is presented in Figure 1. In this approach, the channel-user relevance matrix ($Mrel$) is based on an approximation of time spent in channels based on read and write activity. As not all users are active in all channels, the matrix built based on this approximation is incomplete. The goal of the algorithm is to fill the whole matrix and use the new relevant scores for ranking of recommendations.

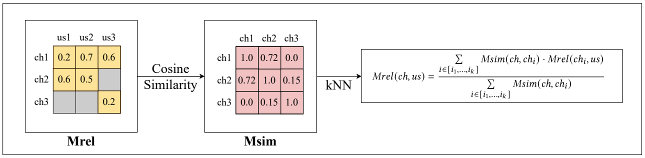

us1
us2
us3
ch1
ch2
ch3
0.2
0.7
0.6
0.6
0.5
0.2
Mrel
Cosine Similarity
1.0
0.72
0.0
0.72
1.0
0.15
0.0
0.15
1.0
Msim
kNN
$Mrel(ch, us) = \frac{\sum_{i \in [i_1,\dots,i_k]} Msim(ch, ch_i) \cdot Mrel(ch_i, us)}{\sum_{i \in [i_1,\dots,i_k]} Msim(ch, ch_i)}$

Figure 1: Slack algorithm pipeline.

First of all, the Slack algorithm builds a similarity matrix for channels ($Msim$) that contains cosine similarity scores for respective channel representations. These representations are the rows in the channel-user relevance matrix that are filled with zeros for unknown values.

$$Msim(ch_1, ch_2) = \frac{C_1 \times C_2}{||C_1|| \cdot ||C_2||}$$

where C_i — row in channel-user relevance matrix corresponding to ch_i ($Mrel(ch_i)$).

After that, for each channel the algorithm finds the most similar channels based on the similarity score. Using k-neighbours, they fill empty values in the initial matrix $Mrel$ with the average of the similarity scores between the target channel and the channels most similar to it, weighted by their relevance to the target user.

$$Mrel(ch, us) = \frac{\sum_{i \in [i_1,\dots,i_k]} Msim(ch, ch_i) \cdot Mrel(ch_i, us)}{\sum_{i \in [i_1,\dots,i_k]} Msim(ch, ch_i)}$$

where $[i_1, .., i_k]$ is a set of indices of the k channels most similar to the target.

In this work, we implement the Slack algorithm to use it as a baseline.

3 METHODOLOGY

3.1 Research Questions

We build this study around two following research questions:

- **RQ1:** How does combining different types of data influence the performance of channel recommendation algorithms?
- **RQ2:** Do users find the suggestions generated by recommendation algorithms valuable?

To answer these questions, we collect the data of three different types (channels, repositories, organizational structure) from the internal systems of JetBrains, a medium-sized software development company. After that, we train several unimodal recommendation algorithms and one unifying multimodal algorithm, and conduct two types of experiments. First, we test the performance of the algorithms on historical data (Section 5.2.1) to find the best unimodal algorithms. After that, we combine the statistics that power these algorithms in a multimodal recommendation model. Finally, to determine the best of unimodal and the multimodal methods, we conduct an assessment of the generated recommendations by the users — employers of JetBrains (Section 5.3).

3.2 Data

Channel activity data is instrumental for the channel recommendation task. In addition, we decided to include data on organizational structure, since this information is available for each employee from the beginning of their work at the company. Finally, since JetBrains is a software company, contributions to source code are a significant part of work for most of employees, so we include VCS history as well. In this section, we describe each of the modalities and discuss challenges with mining and processing the data.

3.2.1 Overview of the data modalities.

Channels. The first data modality that we consider is channels. Each channel is a set of messages. Some messages are grouped

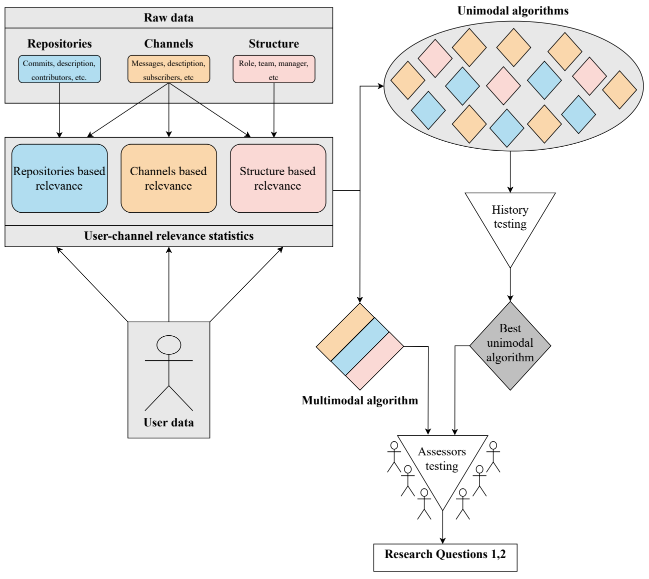
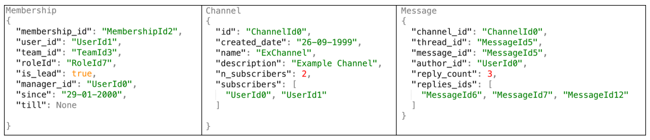


Figure 2: Overview of this study.

together in threads. Additionally, a list of channel subscribers, name, and (optionally) description are available for every channel. This information is the basis for the channel recommendation model.

Organizational Structure. The next data modality is the structure of the organization — team memberships and roles. It is represented by the role of each employee in the company (e.g. "senior researcher"), the identifier of their manager and team, and information about whether the employee is a formal lead of their team. An employee can have more than one membership, e.g. someone can have the role of Researcher in team A and the role of Team Lead in team B. Information about user's memberships in the company can yield recommendations which are focused on the similarity of the target user's and channel subscribers' positions in the company hierarchy, thus possibly mitigating the cold start problem.

Technical repositories. The final modality is the history of version control repositories. For each repository we retrieve its name, description, list of contributors, and commit history. This information provides an understanding of the professional interests and expertise of users and historical data on their collaboration with colleagues that may not be reflected in other modalities. This modality, too, may assist with the diversity of recommendations.



```
Membership
{
  "membership_id": "MembershipId2",
  "user_id": "UserId1",
  "team_id": "TeamId3",
  "roleId": "RoleId7",
  "is_lead": true,
  "manager_id": "UserId0",
  "since": "29-01-2000",
  "till": None

}
```

```
Channel
{
  "id": "ChannelId0",
  "created_date": "26-09-1999",
  "name": "ExChannel",
  "description": "Example Channel",
  "n_subscribers": 2,
  "subscribers": [
    "UserId0", "UserId1"
  ]

}
```

```
Message
{
  "channel_id": "ChannelId0",
  "thread_id": "MessageId5",
  "message_id": "MessageId5",
  "author_id": "UserId0",
  "reply_count": 3,
  "replies_ids": [
    "MessageId6", "MessageId7", "MessageId12"
  ]
}
```

Figure 3: Examples of collected data.

3.2.2 Data sources. The data is obtained from three systems for collaborative software development:

(1) **Space**, a unified platform for the entire software development pipeline and team collaboration. For this study, this system is a source of organizational structure information, channel activity data, and version control repositories;
(2) **Slack**, a business communication platform [10]. Slack is an additional source of channel activity data in our study;
(3) **GitHub**, a source code management system that features issue tracking, continuous integration, and code review [36]. GitHub is an additional source of version control data.

All these systems contain personal data of developers and sensitive information internal to the company. Processing this data is associated with the following risks and challenges.

Privacy concerns. A business communication platform may contain not only work-related information, but also the users' personal information and discussions. To minimize the privacy risk, we only use public data, i.e. the data that is accessible to anyone in the organization, in this study.

Information security. To minimize the risk of unauthorized access to information, we use the absolute minimum of data that is required for the study. For example, we do not access or store the source code of private projects, and we do not store the contents of blobs in the repositories. We process all data in a secure cloud environment with minimal access rights: the authors of this study did not even have a technical ability to download raw data. Finally, all data processing pipelines were reviewed by information security experts.

Ethical concerns. Records of developers' activity can potentially be used to analyze productivity and other work qualities of the developers [37]. Using data to assess productivity can potentially harm the employees. This work is not dedicated to any personal qualities or productivity of developers. To minimize the ethical risks and prevent potential biases, we only consider quantitative information about threads and messages. We do not process or store the contents of threads and messages (e.g. text or attachments).

3.2.3 Data mining. After collecting data from the different sources, we link and filter it for further use. Table 1 presents some general statistics about the collected data of each modality.

Channel activity. There are two sources of channel activity data in our study: Slack and Space. For Slack, we use its built-in functionality to download an archive of public channels [38]. It contains messages for each channel and general channel statistics: the number of messages and subscribers. For Space, we use the API to retrieve the channel activity data. For each message we store its author and timestamp. Additionally, a list of channel subscribers, its name and description are available.

To combine the data from the two sources, we match users' accounts via their email addresses. Also, we filter out the channels that were inactive for the last three months and channels with less than three subscribers.

Organizational structure. The source of information about the organizational structure is Space. The organizational structure in Space is represented with memberships. Each membership includes the manager, the team and the role. For each user, a history of previous roles in the company is also available. In addition, users' known emails are available to retrieve via an API. Space is a single source of information about the organizational structure in our study, so there is no need to combine this data with other systems, like we do for other modalities.

Table 1: Data description

Organization's Structure	
Number of employees	~ 2000
Number of teams	~300
Average number of employees in a team	9
Channels	
Number of Space channels	~ 150
Average number of Space subscribers	45
Number of Slack channels	~ 300
Average number of Slack subscribers	88
Historical depth	6 months
Technical Repositories	
Number of Space repositories	~ 2000
Average number of Space contributors	10
Number of GitHub repositories	~ 600
Average number of GitHub contributors	20

Technical repositories. There are two sources of the version control repositories: GitHub and Space. Information about projects and repositories hosted in Space is available via the API. All information from GitHub is available via its HTTP API [39].

To match Space and GitHub data, we use users' emails specified in both platforms. One issue with mining the repositories data was that some repositories existed both at Space and GitHub as mirrors. To remove such duplicates, we compared the hashes of recent commits for all pairs of repositories and filtered out the ones with at least one matching commit.

Figure 4 presents an overview of the data collection and processing pipeline. Figure 3 shows examples of Json files with some of the collected data.

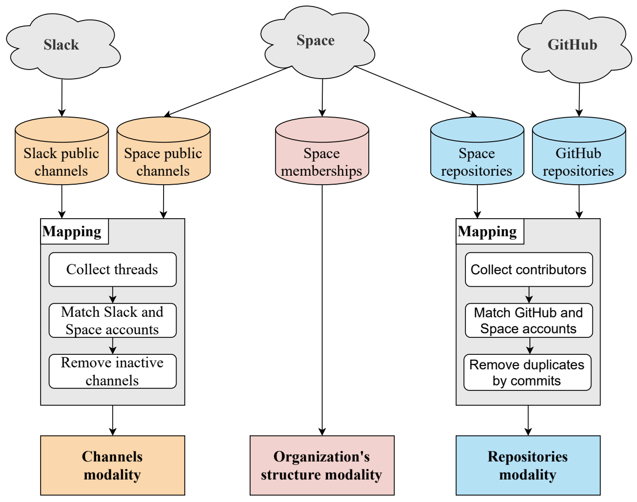


Figure 4: Data pipeline.

3.3 Experiment setup

To answer the research questions, we build several models based strictly on channel activity data and related statistics. We also build several models that are based on statistics from another modality (repositories or structure). However, these models also use information about channel subscribers, which is essential in the channel recommendation task. Since these methods mostly rely on statistics from one modality, we still refer to them as *unimodal* methods. Finally, we use the statistics collected for all three modalities to build a multimodal channel recommendation system. We detail the statistics and the recommendation methods in in Section 4.

We use two types of experiments to evaluate constructed models: history testing and user assessment. We use history testing during the training process to get a preliminary evaluation of the models, and also to select the best models for further analysis. To make conclusions on the systems' performance and to answer research questions, we conduct an assessment of the recommendations by the users.

3.3.1 History Testing. For historical evaluation, we split the whole dataset into "train" and "test" parts. The test part includes the last 3 channels each user has subscribed to. All other channels are left for training. The goal of the recommendation models is to generate 3 suggestions for each user. The measure of performance is the predictive ability of the model, measured out in precision and ROC AUC. We detail the historical experiment in Section 5.1.1.

3.3.2 User evaluation. The user evaluation is vital to assess the real-world performance of the models. In addition, it shows how well the recommendation algorithms are suited for different types of users (e.g. depending on their role or overall activity in chats).To conduct this experiment, we created a recommendation bot in Space and asked JetBrains employees to use it and assess the recommendations. In total, there were over 70 participants of various different roles and responsibilities (e.g. programmers, technical writers, managers), experience, activity levels (e.g. contributed into 10+ repositories, or 1-3 repositories). We provide the details about the bot and the experiment in Section 5.1.2.

4 CHANNEL RECOMMENDATION

Figure 2 presents a step-by-step overview of this study. After creating the datasets containing all the information about users, channels, repositories and organization's structure, we extracted various statistics (Section 4.1) characterizing relevance of the existing channels to all the users. By using different combinations of these statistics, we build unimodal channel recommendation models we mention in Section 3.3. History testing of these models allowed us to choose the best unimodal algorithms and statistics they were based on.

After training a multimodal model built on the performing best in history testing, we compare this method with the best unimodal algorithm and a random recommender with user assessments (Section 5.3). With a recommender bot we describe in Section 5.1, the users at JetBrains provided us with feedback on quality of the generated channels, which allowed us to answer the research questions (Section 3.1).

In this section, we detail the statistics and the mechanics of the recommendation models.

4.1 Statistics

In this subsection, we describe the statistics collected from the data on channels, repositories, and organizationl structure to be later fed into various channel recommendation algorithms. Notably, all these statistics are not only applicable to channel recommendation problem but can also be reused in other socio-technical assistance methods, as their performance in the recommendation models suggests that they represent the affinity of developers and various entities to each other. For example, one could reuse repositories-related statistics (e.g. *us2rep_col* in Section 4.1.1) in a reviewer recommendation system.

4.1.1 User2Item interaction statistics. User-to-item statistics represent interactions between users and different items (channels or repositories) via relevance matrices. For each item, we select several levels of activity to calculate the statistics on. These levels are logically nested into each other: for example, one cannot become a collaborator to a repository without making changes to some files.

Channels. The channel statistics describe how users interact with channels on each of the three levels of detail: subscription, threads and messages. Channel statistics can be used as factors in some unimodal recommendation systems, such as matrix factorization (Section 4.2.4).

us2ch_sub, subscription-level statistics, represent whether the user *us* is subscribed to the channel *ch*. These statistics only contain 0s and 1s. More formally,

$$us2ch\_sub(us, ch) = us \in F(ch),$$

where $F(ch)$ is the set of subscribers for channel ch.

us2ch_thr, thread-level statistics, represent the fraction of threads in the channel that the user has participated in (i.e. wrote at least one message to the thread).

$$us2ch\_thr\_[date](us, ch) = (us \in F(ch)) \cdot \frac{|T(us,ch,date)|}{|T(ch,date)|},$$

where $T(ch, date)$ is the list of threads in the channel with messages after certain $date$, $T(us, ch, date)$ are threads that user has posted to after certain $date$, $[date]$ is the starting date to consider in the calculation.

us2ch_mes, message-level statistics, represent the fraction g^1 of messages the user has written in each thread of the channel.

$$us2ch\_mes\_[date](us, ch) = (us \in F(ch)) \cdot \frac{G(us,date)}{|T(ch,date)|}$$

$$G(us, date) = \sum_{thr \in T(us,ch,date)} \frac{|Mes(us, thr, date)|}{|Mes(thr, date)|}$$

where $Mes(thr, date)$ are all messages in the thread since $date$, $Mes(us, thr, date)$ are messages that the user wrote to the thread since $date$.

Repositories. We use two statistics representing users' interactions with code repositories on the level of collaboration and individual files. Repository statistics can be further applied as factors in matrix factorization based algorithms by combining *user2rep* scores for all channel subscribers.

us2rep_col, collaboration-level statistics, represent whether a user is one of a repository collaborators, i.e. if they authored any commits in this repository. This statistic only contains 0s and 1s.

$$us2rep\_col(us, rep) = us \in C(rep),$$

where $C(rep)$ is the set of the repository's collaborators.

us2rep_file, file changes level statistics, represent the fraction g^2 of files on each branch of the repository that the user has changed.

$$us2rep\_file(us, rep) = (us \in C(rep)) \cdot \frac{\sum_{br \in Br(us,rep)} g^2(us,br)}{|Br(rep)|}$$

$$g^2(us, br) = \frac{|F(us, br)|}{|F(br)|},$$

where $Br(rep)$ are all branches in the repository, $Br(us, rep)$ are branches that the user has committed to, $F(br)$ are all files on the branch, $F(us, br)$ are files that the user has changed in at least one commit reachable from the branch.

4.1.2 User2User Interactions. User-to-user statistics represent similarities between users based on one of the data modalities: channels, repositories or structure. Similarly to *user2item* statistics, here for each modality we build users' similarity matrices on different levels of interaction. Similarly to how, for example, the *word2vec* embedding model [40] specifies "target" and "context" roles for words, we distinguish between the "target user" and a "colleague". The target user is the one we are going to produce recommendations for. A colleague is some other user whose information on their interactions with items we are going to use to find patterns that might help to make predictions for the target user. We use the user2user statistics to build user-based recommendation systems, such as user aggregation based algorithms (Section 4.2.1).

Channels. The channel statistics represent the similarity of users to each other based on their interaction in channels on three levels, similarly to *user2ch* statistics. Later, these scores are averaged among all the channel followers (as colleagues), to determine the relevance score of the channel to the target user.

us2us_sub, subscription-level statistics, represent the ratio of target user's and colleague's common subscriptions $S(us, col)$ to the number of the target user's channels subscriptions.

$$us2us\_sub(us, col) = \frac{|S(us, col)|}{|S(us)|}$$

$$S(us, col) = S(us) \cap S(col),$$

where $S(us)$ are channels that user is subscribed to.

us2us_thr, thread-level statistics, represent the ratio g^3 of threads in each common channel that both users have participated in $T(us, col, ch)$ to the target user's threads.

$$us2us\_thr\_[date](us, col) = \frac{\sum_{ch \in S(us,col)} g^3(us,col,ch,date)}{|S(us)|}$$

$$g^3(us, col, ch, date) = \frac{|T(us, col, ch, date)|}{|T(us, ch, date)|}$$

$$T(us, col, ch, date) = T(us, ch, date) \cap T(col, ch, date)$$

us2us_mes, message-level statistics, represent the fraction g^4 of messages two users have written in each shared thread of all their common channels.

$$us2us\_mes\_[date](us, col) = \frac{\sum_{ch \in S(us,col)} \frac{G(us,col,date)}{|T(us,ch,date)|}}{|S(us)|}$$

$$G(us, col, date) = \sum_{thr \in T(us,col,ch,date)} g^4(us, col, thr, date)$$

$g^4(us, col, thr, date) = \frac{|Mes(us,thr,date)|+|Mes(col,thr,date)|}{|Mes(thr,date)|}$

Besides these statistics, we collected statistics for the mentions level. It was similar to messages level except only messages with the target user mentioning the colleague were counted. However, this statistic was too sparse and not useful to any of the algorithms, so we did not use it in the final models.

Repositories. These statistics represent users' similarity based on their interactions within repositories on collaboration and commits levels, similarly to *user2rep*. Later, these scores were be averaged among all the channel followers (as colleagues), to find out the relevance score of some channel to the target user.
us2us_col, collaboration-level statistics, represent the ratio of two users' common repositories they have contributed to $R(us, col)$ to the number of the target user's repositories.

$$us2us\_col(us, col) = \frac{|R(us, col)|}{|R(us)|}$$

$$R(us, col) = R(us) \cap R(col),$$

where $R(us)$ are repositories that the user has contributed to.
us2us_file, file-level statistics, represent the fraction of files users both changed at some point on each branch across all their mutual repositories.

$$us2us\_file(us, col) = \frac{\sum_{rep \in R(us,col)} \frac{G(us,col,rep)}{|Br(us,rep)|}}{|R(us)|}$$

$$G(us, col, rep) = \sum_{br \in Br(us,col,rep)} \frac{|F(us,br) \cap F(col,br)|}{|F(us,br)|}$$

$$Br(us, col, rep) = Br(us, rep) \cap Br(col, rep)$$

Organizational Structure. For the org structure modality, we built an additional matrix *positions* that encodes users' similarity based on their manager, role, team and leadership status. Each number in the matrix is a bit mask that represents which parameters the users share. For example, $positions(us, col) = 13$ (1101) means that these users have the same manager and role, they are both team leads in their own teams, and only teams are different.

$$position(us, col) = bit\_manager, bit\_role, bit\_team, bit\_is\_lead$$

Using the *positions* matrix, we create three position similarity matrices with statistics based on one of the first three bytes in the mask. Values of these matrices are 1 if bits in their related positions equal 1, and 0 otherwise:
us2us_role — role-level statistics — whether two users have the same role.

$$us2us\_role(us, col) = positions(us, col) == \#1\#\#$$

us2us_team — team-level statistics — whether two users belong to the same team.

$$us2us\_team(us, col) = positions(us, col) == \#\#1\#$$

us2us_team-role — role+team-level statistics — whether two users are on the same team with similar roles.

$$us2us\_role\_team(us, col) = positions(us, col) == \#11\#$$

Later, these scores are averaged across all channel followers (as colleagues), to find out the relevance score of some channel to the target user.

4.1.3 Other Statistics. In addition to the statistics described above, we collected the following data: *#subscribers* as a number of subscribers in channels, *#subscriptions* as a number of subscriptions for users, *ch_activity_** as channel activities since some date (e.g. ch_activity_2021-09-01), *bm25_rep2ch* and *bm25_ch2rep* as BM25 scores for channels and projects depending on which one is a query.

Channel activity was measured by the number of messages and threads that were written to channels since some specified date: **ch_activity_[thr|mes]_[date]**. For the date we used 1, 3, 6 and 12 months before the point of data collection and evaluation.

Okapi BM25 [41] is a ranking method originally created for web search engines to estimate documents' (web pages) relevance to a query. It is calculated based on frequencies of query words in each document:

$$BM25(D, Q) = \sum_{q \in Q} IDF(q) \cdot \frac{f(q, D) \cdot (k + 1)}{f(q, D) + k \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

$$IDF(q) = \log \frac{N - n(q) + 0.5}{n(q) + 0.5}$$

where D and Q — document and the query, $f(q, D)$ — frequency of the word q in the document D, $avgdl$ — average size of all the documents in the corpus, k and b — free coefficients, N — number of documents in the corpus, $n(q)$ — number of documents with the word q.

We can use repository name as a query, and channels' names combined with their descriptions as documents (**bm25_rep2ch**), or vice versa (**bm25_ch2rep**).

4.2 Generating the recommendations

Algorithms that recommend entities of some type are usually based solely on records of interaction with entities of the same type. For example, collaborative filtering and matrix factorization are two of the most popular approaches to the problem. Another way to rank items (channels, in our case) is to use decision trees for regression to predict a probability of a subscription. We used all these and some other methods to create the unimodal recommendation models. In the end, we joined the best-performing statistics (based on historical testing, Section 5.2.1) to create a multimodal decision trees based algorithm (XGBoost).

4.2.1 User Aggregation. A logical way to measure channel's relevance to a user is to estimate their similarity to this channel's followers. The more alike they are in at least one of the modalities, the bigger the chance are the user is going to find this channel relevant as well. Using the users' similarities (Section 4.1.2), the relevance of a channel to the user is calculated as follows:

$$user - agg\_us2us\_ * (us, ch) = \frac{\sum_{col \in S(ch)} us2us\_ * (us, col)}{|F(ch)|}$$

where $*$ is one of the levels in one of the three modalities described above (e.g. collaboration *us2us_col* in repositories, or teams *us2us_team* in structure), and $F(ch)$ is the list of channel followers. For thread and message statistics level at channel modality, there is also the date parameter which defines since when we consider these activities (e.g. 2021-09-01).

4.2.2 Item Aggregation. Another way to measure channel's relevance to a user is to average the relevance of its subscribers' items to that user. For example, if the target user's relevance to some channel's subscribers' repositories is high, then they might have something to discuss. The relevance of different items to users is calculated as described in Section 4.1.1. We calculate channel's relevance to the target user as follows:

$$item - agg\_us2ch\_*(us, ch) = \frac{\sum_{col \in F(ch)} \frac{\sum_{ch^* \in S(col)} us2ch\_*(us, ch^*)}{|S(col)|}}{|F(ch)|}$$

$$item - agg\_us2rep\_*(us, ch) = \frac{\sum_{col \in F(ch)} \max_{rep \in R(col)} us2rep\_*(us, rep)}{|F(ch)|}$$

where $*$ is one of the levels in one of the three modalities we have described above (e.g. collaboration $us2ch\_mes_{[}date]$ in channels, or $us2rep\_file$ in repositories), $S(us)$ is the list of user's subscriptions, $R(us)$ are repositories that the user has contributed to. We use the maximum score across colleague's repositories instead of average (which we use for channels): even one highly relevant repository for the target user might mean that they have a common interest. However, for channels, the discussions inside even one channel are often more general and diverse, so we use the average there.

4.2.3 User Based Collaborative Filtering. This is a standard recommendation algorithm, logically similar to the user aggregation described above. However, in this case, the averaging is focused on the most similar users to the target user, instead of all channel subscribers.

$$user - cf\_us2ch\_*(us, ch) = \frac{\sum_{col \in N(us)} us2us\_*(us, col) \cdot (col \in F(ch))}{\sum_{col \in N(us)} us2us\_*(us, col)}$$

$$N(us) = top\_5\{us2us\_*(us, col)\}_{col \in \{U \setminus us\}}$$

where U is the list of all the users.

4.2.4 Matrix Factorization. Embeddings are numerical vector representations of some encoded items. They provide an easy way to measure similarity and rank items. For our task, we can use matrix factorization methods to decompose user/channel similarity matrices ($us2ch\_*$) into latent representations of users and channels. Cosine similarity of those representations gives us an estimate of the relevance score of the respective channel to the user. In this study we tried three embedding models: Bayesian Personalized Ranking (bpr) [42], Alternating Least Squares (als) [43], Logistic Matrix Factorization (lmf) [44]. However, in Section 5.2.1 we only present results for the *bpr* method, since it showed the best performance of these three matrix factorization methods.

$$bpr - us2ch\_*(us, ch) = cos(emb(us), emb(ch))$$

where $*$ is one of the channel levels (subscription, threads, messages), emb is the latent vector representation of an item calculated based on a user2channel relevance matrix $us2ch_*$, cos is the cosine similarity function.

4.2.5 Slack. In 2016, Slack posted an article [13] that described their channel recommendations system, which was a variation of item-based collaborative filtering. The idea was to use kNN regression to fill in missing values in the user-channel relevance matrix. We describe this algorithm in detail in Section 2.3.2. Since the data that we use does not contain any time metrics, we employ users' activities in terms of messages: $us2ch\_thr\_[date]$ and $us2ch\_mes\_[date]$.

$$Msim(ch_1, ch_2) = \frac{us2ch\_*(ch_1) \times us2ch\_*(ch_2)}{||us2ch\_*(ch_1)|| \cdot ||us2ch\_*(ch_2)||}$$

$$slack - us2ch\_*(ch, us) = \frac{\sum_{i \in [i_1, \ldots, i_k]} Msim(ch, ch_i) \cdot us2ch\_*(us, ch_i)}{\sum_{i \in [i_1, \ldots, i_k]} Msim(ch, ch_i)}$$

where $us2ch\_*(ch_i)$ is the row in channel-user relevance matrix $us2ch\_*$ corresponding to ch_i.

4.2.6 XGBoost. XGBoost [45] is a software library that provides a framework regularizing gradient boosting decision trees. In our study, we use it to combine several statistics from different modalities, and train a multimodal recommendation model. We also train a model only based on channel-related scores (e.g. $item - agg_sub$) to make sure that the multimodal model performs better than unimodal algorithms because of additional modalities, and not because it applies several channel statistics. We present the results in Section 5.2.2 and Section 5.3.

5 EVALUATION

5.1 Evaluation methods

For this study, we collected a unique dataset combining several types of data about organization's employees and their interactions.

To train the recommendation systems, we only use channels with (1) with at least 3 subscribers, (2) that were active in the last 6 months, meaning there was at least one message sent to the channel. This was done to filter out "dead" channels. After filtering process, there were about 450 channels left (Table 1).

5.1.1 History testing. To evaluate and compare the recommenders, we use a history testing method with Precision@3, MAP (Mean Average Precision), and ROC-AUC metrics. The task of each recommendation system is to make **3 suggestions** of channels for each user. We chose this number for several reasons:

- Less than 3 recommendations leaves little choice to the users.
- Providing more recommendations considerably decreases the size of the testing dataset: there are about 150 channels in Space.
- More than 3 recommendations means lower relevance on average and less trust in the system. It also generates a longer combined list with higher positional bias (first recommendations are considered more relevant).

Thus, for history testing, all employees subscribed to less than 3 channels were filtered out of the dataset.

In historical testing, we call the channels that the target user is subscribed to "positives", and others are "negatives". The testing dataset contained 3 last Space positives that the user subscribed to (to match the 3 suggestions made by the model), and 30% of Space negatives. All remaining Space positives were joined with all Slack positives, and 70% of Space negatives were joined with all of Slack negatives into a training set.

5.1.2 Assessment by users. Besides history testing, we asked JetBrains employees to evaluate recommendations generated by four models. At this stage, we generated 2 recommendations from each of the following models:
(1) **random**: suggestions for each user are random channels among all available. We use this algorithm to estimate how critical users are, whether they are mostly happy with any suggestions, or they are more thorough about their subscriptions.
(2) **bpr-us2ch_sub**: the best unimodal system that is based solely on channels' data. It represents a big segment of existing approaches that only use the data directly related to the task objective.
(3) **XGB_ch**: decision trees trained on three channel statistics showing the highest performance on historical testing: *#subscribers, bpr-us2ch_sub, item_agg-us2ch_thr_2021-09-01.* We built this model to make sure that multimodal XGBoost model performs better due to the additional modalities, and not the united channel statistics.
(4) **XGB_all**: decision trees trained on the best statistics from all three data modalities: *#subscribers, bpr-us2ch_sub, item_agg-us2ch_thr_2021-09-01, user_agg-us2us_team, user_agg-us2us_team-role, user_agg-us2us_col, item_agg-us2pr_file.* We compare this approach to the unimodal systems to see if (and how) additional modalities improve generated recommendations.

Each of these models produced 2 recommendations, so each survey participant received up to 8 channels to rate (less if there were intersections).

Also, we created an internal channel recommender bot in Space. The bot sends out the recommendations to the users on their request. After getting the recommendations, the user could follow the links to inspect the channels, and rate the recommendations by clicking one of the colored emojis (green, yellow, red) representing the scores of 1, 0, and -1. We asked the users to use the following scores:

- -1 (red), if the channel is irrelevant for the user;
- 0 (yellow), if the channel is somehow relevant or interesting for the user but they would not subscribe to it;
- 1 (green), if the channel is relevant, and the user would subscribe to it.

5.2 Evaluation results

5.2.1 History testing: experiments within modalities. The first stage of evaluation was historical testing. In historical testing, we split the dataset by randomly choosing 30% of negative examples and 3 last positive examples for a testing set, and everything else for the training set. All models are trained on the training set, and then their performance on the testing set is assessed.

Historical testing scores for all models are presented in Table 2. The basic *topk_subscribers* model performed rather well with 61% ROC AUC and 30% MAP. There are not that many active channels in Space yet (Table 1), so the biggest channels are quite likely to be the most general and important ones that are relevant to many users. The best-performing unimodal channels based algorithm was *bpr-us2ch_sub* with the highest ROC-AUC score across all models. It is interesting to see that the best unimodal algorithms based on repositories (*item_agg-us2pr_file*) or structure (*user_agg-us2us_team*) got scores that are close to the *bpr* model. Moreover, Precision@3 and MAP for the unimodal structural algorithm were even higher than for the unimodal channel algorithm.

Table 2: Results of historical testing: unimodal algorithms.

Algorithm	Pr@3 %	MAP %	ROCAUC %
random	8	10	50
topk_subscribers	35	30	61
topk_bm25-pr2ch	18	17	58
topk_bm25-ch2pr	18	18	56
slack-us2ch_thr_2021-09-01	19	16	53
slack-us2ch_mes_2021-09-01	19	16	54
user_agg-us2us_sub	24	18	54
user_agg-us2us_thr_2021-09-01	29	24	68
user_agg-us2us_mes_2021-09-01	32	25	69
user_agg-us2us_col	32	26	67
user_agg-us2us_file	34	29	66
user_agg-us2us_team	**<u>42</u>**	**<u>33</u>**	69
user_agg-us2us_role	17	16	57
user_agg-us2us_team-role	35	28	66
item_agg-us2ch_thr_2021-09-01	35	25	74
item_agg-us2ch_mes_2021-09-01	35	26	74
item_agg-us2pr_col	31	25	67
item_agg-us2pr_file	34	29	67
user_cf-us2us_thr_2021-09-01	38	29	66
user_cf-us2us_mes_2021-09-01	37	28	65
bpr-us2ch_sub	40	30	**<u>75</u>**
bpr-us2ch_thr_2021-09-01	29	22	68

Date format is YYYY-MM-DD. We tried several time spans (1, 3, 6, 12 months). Using the data for 3 months before the evaluation yielded the highest performance.

5.2.2 History testing: experiments between modalities. We used the statistics we collected from the unimodal methods to train the XGB_all and XGB_ch models.

XGB_all achieved 82% ROC AUC, which is 7% higher than the best unimodal method ROC AUC. Its precision scores are also considerably higher: 62% Precision@3 and 55% MAP which is 20% and 12% better than the best unimodal algorithms.

XGB_ch achieved 61% Precision@3, 54% MAP and 81% ROC AUC, which is similar to the *XGB_all* multimodal method. Because of the similar performance, we could not compare it with the multimodal model until the evaluation by users.

5.3 Assessment by users

The results of user assessment are presented in Table 3. Feedback from users received during this experiment suggests the following.
(1) On average, channels in Space are mostly irrelevant to users: since 73% of random suggestions were marked irrelevant. Thus, the task of finding the few that are more relevant to the user is valid in this environment.
(2) The unimodal channel-based model (*bpr-us2ch_sub*) is a good baseline to work with. Its recommendations are rated better than those of the random one: over 25% less irrelevant suggestions, and 10% more subscriptions.
(3) The XGB_ch model shows a decrease in irrelevant recommendations, and increase in subscriptions, relative to *bpr-us2ch_sub.*

Table 3: User assessment results.

Method	-1, %	0, %	1, %
random	73%	16%	11%
bpr-us2ch_sub	46%	33%	21%
XGB_ch	**27%**	**42%**	31%
XGB_all	32%	22%	**46%**

Columns represent users' ratings of the channels: (-1) is irrelevant, (0) is relevant but not subscription worthy, (1) is subscription worthy.

(4) The XGB_all model provides slightly higher percentage of irrelevant recommendations, however, its number of subscriptions is significantly higher: It is 15% more than the best score in the XGB_ch model. Since providing relevant recommendations is the goal for the models, XGB_all shows the best results among all the compared channel recommendation algorithms.

6 THREATS TO VALIDITY

There are several threats to validity of this study.

6.0.1 Internal validity. The platform we used to conduct the experiments (Space) is relatively new, and its user base at JetBrains is still growing. At the time of the experiment, the number of active channels was relatively low (Table 1). We could notice the intersection in recommendation sets for different people, which makes the recommendation task less personalized in this context. Also, we did not analyze the performance of the algorithms on employees with different roles (e.g. software engineers, designers, HR). Employees with different roles might use different modalities a lot less or more often (e.g. HR probably do not have technical repository data). This is worth studying in future work.

6.0.2 External validity. Our study was conducted in an organization with a specific field of work. Most of the channels in the workspace of JetBrains are dedicated to specific topics that are mostly only interesting for the limited number of people. Besides, people from different teams often work on related tasks. For these two reasons, some users may receive recommendations of more general channels, rather than of those specific to their personal interests. Bigger and more general channels of different teams (e.g. a channel with news about a new project) turned out to be interesting for many users, even if they were not directly involved with the topic. This situation might be less likely possible in companies with more diverse scopes of work, such as outsourcing companies.

Finally, the need for a channel recommendation system might simply not be present in some organizations. If the organization is too small, or if it has some external systems augmenting the discovery of internal communication resources (e.g. map of channels), our results are simply not relevant.

7 RESULTS AND CONCLUSION

RQ1: By combining three types of data — channels, repositories, and organization structure — we can improve the quality of recommendation algorithms in terms of perceived relevance. The model based on three types of data generated the same amount of irrelevant recommendations as the best of unimodal algorithms, yet the amount of "subscription worthy" recommendations was significantly higher (Section 5.3).

Besides that, the best unimodal models based on repository activity or organizational structure show score quite similar to the best unimodal channel-based models. This leads us to believe that the channel data, technical, structural data are equally important for our task. This is important, because channel data (even public) is harder to acquire and work with due to privacy and ethical issues.

Structural data appears to be particularly useful for the new users who do not have any channel subscriptions or repository contributions yet. So, a specific study on how different modalities handle new users is an interesting direction for future work.

Since including different types of data into the channel recommendation algorithms improves their performance, we believe that it might also be true for other socio-technical assistance algorithms, such as reviewer recommendation systems. This is the idea we are going to study in our future work, along with including other modalities, such as meetings, code review discussions, and issue tracking records.

RQ2: Assessors only found less than a third of the suggestions generated by the best recommendation algorithms to be irrelevant. In the user evaluation of the multimodal model, only 30% of recommendations was marked as irrelevant, compared for over 70% for random recommendations. Besides that, almost half of the recommendations generated by multimodal model were marked as "subscription worthy". This suggests that channel recommendation systems are capable of helping users find interesting channels they did not know about, which could promote information exchange and ultimately support the organization.

7.1 Conclusion

To our knowledge, this study is the first on the effects of multimodal data on channel recommendation algorithms. By testing multiple algorithms based on different types of data, we explore how modalities influence quality of recommendations. Given the ongoing evolution of developer tools into multifaceted platforms and our results, we are calling for more work on multimodal approaches to RSSE problems.

7.2 Reproducibility

Due to trade secrecy and privacy law, we keep the data collection code and the dataset private. The code for calculation of the statistics, implementation of the models, and historical testing is available on Zenodo[2].

8 ACKNOWLEDGEMENTS

We thank our colleagues at JetBrains for their help and cooperation. Special thanks to the Space team, Alexey Sviridov, Klara Salamounova, Yaroslav Russkih, Alexander Syplyvchak, and Vadim Reider.

[2]Reproduction package: https://doi.org/10.5281/zenodo.5889598

REFERENCES

[1] S. Overflow. (2021) 2021 Stack Overflow developer survey. [Online]. Available: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-other-tools

[2] D. Lahat, T. Adali, and C. Jutten, "Multimodal data fusion: An overview of methods, challenges, and prospects," *Proceedings of the IEEE*, vol. 103, no. 9, pp. 1449–1477, 2015.

[3] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *2013 10th Working Conference on Mining Software Repositories (MSR).* IEEE, 2013, pp. 22–30.

[4] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K.-i. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 141–150.

[5] I. Avazpour, T. Pitakrat, L. Grunske, J. Grundy, M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, "Recommendation systems in software engineering," *Dimensions and metrics for evaluating recommendation systems*, pp. 245–273, 2014.

[6] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, p. 712–721.

[7] E. Mirsaeedi and P. C. Rigby, "Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20, 2020, p. 1183–1195.

[8] A. Chueshev, J. Lawall, R. Bendraou, and T. Ziadi, "Expanding the number of reviewers in open-source projects by recommending appropriate developers," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 499–510.

[9] M. Bowler. (2005) Truck factor. [Online]. Available: http://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/

[10] "Slack," https://slack.com/, accessed: 2022-01-18.

[11] Slack. Channels: Discover a new way of working. Accessed: 2022-01-11. [Online]. Available: https://slack.com/features/channels

[12] ——. (2019) How channels extend the reach of internal communications. [Online]. Available: https://slack.com/blog/collaboration/slack-on-slack-how-channels-extend-the-reach-of-internal-communications?from=channels

[13] "Personalized channel recommendations in Slack," https://slack.engineering/personalized-channel-recommendations-in-slack/, accessed: 2022-01-18.

[14] H. A. Çetin, E. Doğan, and E. Tüzün, "A review of code reviewer recommendation studies: Challenges and future directions," *Science of Computer Programming*, vol. 208, p. 102652, 2021.

[15] "JetBrains Space," https://www.jetbrains.com/space/, accessed: 2022-01-18.

[16] M. Gasparic and A. Janes, "What recommendation systems for software engineering recommend: A systematic literature review," *Journal of Systems and Software*, vol. 113, pp. 101–113, 2016.

[17] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE software*, vol. 27, no. 4, pp. 80–86, 2009.

[18] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra, "Aroma: Code recommendation via structural code search," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–28, 2019.

[19] V. Raychev, M. Vechev, and E. Yahav, "Code completion with statistical language models," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 419–428.

[20] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *2009 6th IEEE International Working Conference on Mining Software Repositories*, 2009, pp. 131–140.

[21] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *2013 20th Working conference on reverse engineering (WCRE).* IEEE, 2013, pp. 182–191.

[22] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*, 2012, pp. 85–89.

[23] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2275–2292, 2012.

[24] B. Alkhazi, A. DiStasi, W. Aljedaani, H. Alrubaye, X. Ye, and M. W. Mkaouer, "Learning to rank developers for bug report assignment," *Applied Soft Computing*, vol. 95, p. 106667, 2020.

[25] V. Kovalenko, N. Tintarev, E. Pasynkov, C. Bird, and A. Bacchelli, "Does reviewer recommendation help developers?" *IEEE Transactions on Software Engineering*, vol. 46, no. 7, pp. 710–731, 2018.

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

[27] E. Sülün, E. Tüzün, and U. Doğrusöz, "Rstrace+: Reviewer suggestion using software artifact traceability graphs," *Information and Software Technology*, vol. 130, p. 106455, 2021.

[28] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 375–384.

[29] R. Romero, E. Parra, and S. Haiduc, "Experiences building an answer bot for gitter," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 66–70.

[30] E. Shihab, Z. M. Jiang, and A. E. Hassan, "Studying the use of developer irc meetings in open source projects," in *2009 IEEE International Conference on Software Maintenance.* IEEE, 2009, pp. 147–156.

[31] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N. A. Kraft, "Exploratory study of slack q&a chats as a mining source for software engineering tools," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR).* IEEE, 2019, pp. 490–501.

[32] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik, "Why developers are slacking off: Understanding how software teams use slack," in *Proceedings of the 19th acm conference on computer supported cooperative work and social computing companion*, 2016, pp. 333–336.

[33] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining.* Ieee, 2008, pp. 263–272.

[34] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, "Addressing cold-start problem in recommendation systems," in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, 2008, pp. 208–211.

[35] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

[36] "GitHub," https://github.com/, accessed: 2022-01-18.

[37] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive?: empirical study of affectiveness vs. issue fixing time," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015, pp. 303–313.

[38] "Guide to Slack import and export tools," https://slack.com/help/articles/204897248-Guide-to-Slack-import-and-export-tools, accessed: 2022-01-18.

[39] "GitHub REST API," https://docs.github.com/en/rest, accessed: 2022-01-18.

[40] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13, 2013, p. 3111–3119.

[41] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, pp. 333–389, 2009.

[42] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, ser. UAI '09. AUAI Press, 2009, p. 452–461.

[43] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 263–272.

[44] C. C. Johnson, "Logistic matrix factorization for implicit feedback data," *Advances in Neural Information Processing Systems*, vol. 27, no. 78, pp. 1–9, 2014.

[45] "Get Strated with XGBoost," https://xgboost.readthedocs.io/en/stable/get_started.html, accessed: 2022-01-18.