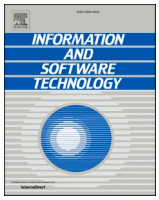
Information and Software Technology 129 (2021) 106394

Contents lists available at ScienceDirect

Information and Software Technology

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



Recommending tags for pull requests in GitHub

Jing Jiang [a], Qiudi Wu [a], Jin Cao [a], Xin Xia [b], Li Zhang [*,a]

[a] State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
[b] Information Technology, Monash University, Melbourne, VIC, Australia

ARTICLE INFO

Keywords:
Tag recommendation
Pull request
Open-source project
Github

ABSTRACT

Context: In GitHub, contributors make code changes, then create and submit pull requests to projects. Tags are a simple and effective way to attach additional information to pull requests and facilitate their organization. However, little effort has been devoted to study pull requests' tags in GitHub.
Objective: Our objective in this paper is to propose an approach which automatically recommends tags for pull requests in GitHub.
Method: We make a survey on the usage of tags in pull requests. Survey results show that tags are useful for developers to track, search or classify pull requests. But some respondents think that it is difficult to choose right tags and keep consistency of tags. 60.61% of respondents think that a tag recommendation tool is useful. In order to help developers choose tags, we propose a method FNNRec which uses feed-forward neural network to analyze titles, description, file paths and contributors.
Results: We evaluate the effectiveness of FNNRec on 10 projects containing 68,497 tagged pull requests. The experimental results show that on average, FNNRec outperforms approach TagDeepRec and TagMulRec by 62.985% and 24.953% in terms of $F1 - score@3$, respectively.
Conclusion: FNNRec is useful to find appropriate tags and improve tag setting process in GitHub.

1. Introduction

Various open-source software hosting sites, notably Github, provide support for pull-based development and allow developers to make contributions flexibly and efficiently [1]. In GitHub, contributors make code changes, then create and submit pull requests to projects [2]. Then members of the project's core team (from here on, integrators) inspect pull requests, and decide whether to accept pull requests and merge modified code [1]. A common way to facilitate the organization of pull requests in projects is based on the use of tags[1]. According to pull requests' information, integrators assign tags to some pull requests from tag library. Tags are a simple and effective way to attach additional information (e.g., metadata) to pull requests [3]. However, tags are sometimes neglected by integrators. For example, in our dataset which contains 112,705 pull requests, 39.22% of pull requests do not have any tags.

In this paper, we conduct a survey to understand usage of tags in GitHub. Survey results show that tags are used to describe functions, priorities, statuses and components, which helps developers to track, search or classify pull requests. However, some respondents think that it is difficult to choose right tags and keep consistency of tags. Meanwhile, it is time-consuming to select tags from the tag library. In order to solve these problems, we further ask respondents' attitude towards a tag recommendation tool. 60.61% of respondents think that a tag recommendation tool is useful. In previous work [4], developers also suggested desired features of bots, such as automatically labeling issues. Therefore, an automatic tag recommendation approach is required to assign tags to pull requests.

There have been several studies [5–10] about tag recommendation in software information sites, such as StackOverflow and Freecode. Zhou et al. proposed a new software object multi-classification method TagMulRec which recommended tags for large-scale evolving software information sites [8]. However, previous works are mainly designed for software information sites, and it remains unknown whether these approaches are effective to recommend tags in GitHub. Pull requests are used to submit code, and have special information such as code file

* Corresponding author.
E-mail addresses: jiangjing@buaa.edu.cn (J. Jiang), 1040814720@qq.com (Q. Wu), 13277061183@163.com (J. Cao), xin.xia@monash.edu (X. Xia), lily@buaa.edu.cn (L. Zhang).

[1] https://help.github.com/articles/labeling-issues-and-pull-requests/

https://doi.org/10.1016/j.infsof.2020.106394
Received 9 September 2019; Received in revised form 14 July 2020; Accepted 17 August 2020
Available online 7 September 2020
0950-5849/

paths.

Respondents in our survey also mention that a recommendation tool should consider text, code and history information. According to these suggestions, we mainly consider attributes, including titles, description, file paths and contributors. Each tag can be considered as a category, and thus tag recommendation is mapped to a multi-label classification problem, which classify pull requests to appropriate categories. The feed-forward neural network is widely used in classification task, and it does not rely on human-engineered features to make classification [11]. We propose a method FNNRec which uses **F**eed-forward **N**eural **N**etwork to recommend tags for pull requests in GitHub projects.

In an effort to demonstrate the effectiveness of our approach, we collected datasets from GitHub. In total, we analyze 10 projects and 68,497 tagged pull requests. TagDeepRec [10] and TagMulRec [8] are originally designed to recommended tags for large-scale evolving software information sites. In comparison, we adopt TagDeepRec and TagMulRec to analyze pull requests' titles and descriptions, and recommend tags for pull requests. We measure the performance of approaches in terms of precisions, recalls and F1-scores. The experimental results show that on average across 10 projects, FNNRec outperforms approaches TagDeepRec [10] and TagMulRec [8] by 62.985% and 24.953% in terms of $F1-score@3$, respectively.

The main contributions of this paper are as follows:

- We make a survey on the usage of tags in pull requests. Survey results show that tags are useful for developers to track, search or classify pull requests. However, it is difficult to choose right tags and keep consistency of tags. 60.61% of respondents think that a tag recommendation tool is useful.
- In order to recommend tags, we propose a method FNNRec which uses feed-forward neural network to analyze titles, description, file paths and contributors.
- We evaluate FNNRec based on a broad range of datasets. Results show that FNNRec outperforms approaches TagDeepRec [10] and TagMulRec [8] by substantial margins.

The reminder of the paper is organized as follow. Section 2 presents the process of setting tags, data collection and statistics. Section 3 presents our survey about the usage of tags in pull requests. Section 4 presents our tag recommendation approach FNNRec. Section 5 presents an empirical evaluation of the approach. Section 6 discusses threats to validity, and Section 7 discusses related works. Finally, Section 8 concludes this paper.

2. Background and data collection

In this section, we begin by providing background information about the process of setting tags in GitHub. Then, we introduce how our datasets are collected, and report statistics of our datasets.

2.1. The process of setting tags

GitHub is a web-based hosting service for software development repositories [12]. In GitHub, contributors make their code changes independent of one another. When a set of changes is ready, contributors create and submit pull requests to projects. Titles and description are written to introduce pull requests, and modified file paths are also shown in pull requests [13]. According to pull requests' information, some integrators assign tags to some pull requests from tag library. In GitHub, only integrators with write access can assign tags to pull requests. If developers do not have write access, they cannot assign tags to their own pull requests. However, tags are sometimes neglected by integrators. For example, in our dataset which contains 112,705 pull requests, 39.22% of pull requests do not have any tags.

To illustrate the contribution process, Fig. 1 shows an example of a pull request with number 21,481 in project *ceph*[2]. We only show part of characters in developers' names, so as to protect developers' privacy. A contributor *ba**** modified code and submitted a pull request. The pull request's title was "common: silence compiler warning", and its body was "Fixes: http://tracker.ceph.com/issues/23774 Signed-off-by: Pa*** Do*** pd***@redhat.com". Then tags ``*bugfix*'', ``*common*'', and "*needs − review*" were chosen from tag library, and assigned to this pull request.

2.2. Data collection and statistics

GitHub provides access to its internal data through an API. It allows us to access rich collection of open-source software projects, and provides valuable opportunities for research. We gather information through GitHub API and create datasets of projects.

In data collection, we choose popular projects, because they receive many pull requests and provide enough information for experiments. We obtain a list of projects from previous work [14], which made their research projects public[3]. We sort their projects by the number of pull requests, and obtain 100 projects with the highest number of pull requests.

We collected pull requests of these 100 projects through GitHub API in June 2017. We sent queries to GitHub API, received its replies, and extracted data from project creation time to June, 2017. We collected pull requests' identifiers, tags, contributors, the creation time, the close time and paths of modified files. Contributors wrote titles and description to summarize the modification of a pull request, which were also gathered.

Some pull requests have tags, while others do not have tags. We select projects with more than 3000 tagged pull requests, which provide enough datasets for experiments. Next, we choose projects with greater than or equal to 30 tags in their tag libraries. If projects have few candidate tags, it is easy to manually assign tags to pull requests. Finally, we obtain 10 projects which satisfy above requirements. Table 1 presents statistics of 10 projects. The columns correspond to project owner (Owner) and name (Project), the number of pull requests (# Pull requests), the number of tags in the tag library (# Tags in the tag library), the number of pull requests with tags (# Tagged Pull requests), and the average number of tags per pull request (Average # tags per pull request). In total, our datasets include 68,497 tagged pull requests and 902 tags. 4 projects have more than 2 tags per pull request, while the average number of tags is between 1 and 2 in 6 projects. Our datasets and code are publicly available, and they can be downloaded from the project homepage[4].

3. Survey on tags

Previous work [3] quantitatively analyzed the use of tags in issues. They found that using labels favored the resolution of issues. In GitHub, developers write issue reports to identify bugs and document feature requests, while developers submit pull requests when they want to merge code changes into main repositories [15]. In this section, we conduct a survey to understand tags in pull requests. More specifically, we design a survey to includes 6 questions.

1. *What is the usage of tags? Would you please list some categories of tags?*
2. *What are benefits of setting tags for the pull requests?*
3. *What are difficulties in setting tags for pull requests?*

[2] https://github.com/ceph/ceph/pull/21481
[3] https://github.com/Yuyue/pullreq_ci/blob/master/all_projects.csv
[4] https://github.com/wqdbuaa/Label-recommendation

Fig. 1. An example of tags in a pull request.

Table 1
Basic Statistics of projects.

Owner	Project	# Pull requests	# Tags in tag library	# Tagged pull requests	Average # tags per pull request
ceph	ceph	14,549	47	11,363	2.208
tgstation	tgstation	17,313	42	10,772	1.395
elasticsearch	elasticsearch	11,076	331	9,135	3.566
owncloud	core	11,701	107	7,863	1.891
symfony	symfony	14,022	67	6,355	1.841
rails	rails	18,168	32	6,088	1.208
angular	angular.js	7,284	96	4,963	2.556
RIOT-OS	RIOT	5,324	51	4,702	2.958
pydata	pandas	6,259	94	4,028	1.927
bitcoin	bitcoin	7,009	35	3,248	1.217
Total		112,705	902	68,497	

4. *Will it be useful or useless if there is a tool to recommend tags for pull requests?*
5. *If you choose useful in question 4, what features should be considered in recommendation tool?*
6. 6. *If you choose useless in question 4, why is a label recommendation tool useless?*

Questions 1,2,3,5 and 6 are open-ended. We provide three choices for question 4, including 'Useful', 'Useless' and 'Unsure'. If respondents choose 'Useful', we ask them question 5, and if respondents choose 'Useless', we ask them question 6.

According to Table 1, we randomly select 200 integrators who ever set tags in these 10 projects and provide email addresses. We send them emails with title 'Survey about tags for pull requests', and ask the above questions. We receive responses from 33 developers.

Tag Usage. The first question is about the usage of tags in pull requests. Cabot et al. performed a clustering analysis to aggregate tags in issues and identified 4 categories of issues, including 'priority', 'version', 'workflow' and 'architecture' [3]. In GitHub, some issues may discuss questions, which are solved by modified code submitted in pull requests. Pull requests' tags may be similar to issues' tags. According to categories in previous work [3], the first author reads all replies and builds categories for the usage of tags. The second author also refers to categories in previous work [3], independently reads all 33 responses, and sets up corresponding categories. Finally, two authors discuss their results and agree on the final set of categories. As shown in Table 2, we define 5 categories of usage of tags. Categories 'Give priority', 'Define status' and 'Describe component' correspond to categories 'priority', 'workflow' and 'architecture' in previous work [3].

Table 2
What is the usage of tags?.

Tag usage	Respondents
Mark function	13 / 39.39%
Give priority	11 / 33.33%
Define status	8 / 24.24%
Describe component	7 / 21.21%
Other	7 / 21.21%

Some respondents mention several usages, and they are classified into multiple categories. After completing the manual labeling process, the two authors discuss their disagreements to reach a common decision. Cohen's kappa coefficient is a measure of the agreement between two raters who determine the categories of subjects [16]. Cohen's kappa coefficient is between 0 and 1. 0 means agreement equivalent to chance, and 1 means perfect agreement. We used Cohen's kappa coefficient to measure the agreement between two authors. Cohen's kappa coefficient is 0.92, which shows near-perfect agreement. Some responses are initially classified as 'other' by an author, but they are finally classified as 'Mark function' or 'Give priority' after discussion. Table 2 shows the usage of tags. From the table, we notice that:

1) The most common answer about usage of tags is that integrators use them to mark functions of pull requests. For example, a respondent mentions that "Categorize if something is a bug or feature; related to documentation;"
2) 33.33% of respondents reply that tags can give the priority of pull requests (e.g., high-priority, important or urgent). For example, some respondents mention "Giving priority" or 'severity/importance".
3) 8 respondents think that tags are used to define current status of pull requests. For example, a respondent says that: "not ready" - a pull request is not ready for review. "on hold" - a pull request is blocked due to other priorities. "manual merge" - caution or manual steps are needed to merge this PR.
4) 21.21% of respondents mention the architectural components affected by pull requests. For example, a respondent mention that "Since Ruby on Rails is divided into components, there is a label for each of these components"
5) 7 respondents mention other reasons. For example, a response is "Labels are used to group PR, so related contributor developer can review his/her related tagged PR."

Benefits of Tags. The second question is about benefits of setting tags for the pull requests. We follow the same process that we describe in question 1. Table 3 displays benefits of setting tags for pull requests. From results, we can note

1) 12 respondents mention that setting tags is convenient for developers to track pull requests. For instance, a respondent writes that "keeping track of state".
2) 12 respondents think that setting tags help developers search pull requests. For example, a respondent mentions that "They mostly help to let users find the pull requests they are most interested in."
3) 5 respondents point out that the benefit of setting tags is to classify pull requests. For instance, a respondent writes that "Helps with categorizing PRs"
4) 6 respondents mention other benefits. For example, a respondent says that "Helps developers effectively manage hundreds of issues and pull requests".

Difficulties on Tag Usage. Third, we want to explore difficulties in setting tags for pull requests. Table 4 shows difficulties in setting tags. From results, we can note

1) 8 respondents find it difficult to choose appropriate tags. For instance, a response is "Sometimes it is unclear what labels are appropriate for a particular pull request." An automatic tag recommendation approach can help developers to find appropriate tags.
2) 5 respondents says that they cannot create new tags when all current tags are inappropriate for pull requests. For example, a respondent says that "Our main problem with labels are that developers without push rights can not add labels. This is quite a problem."
3) 4 respondents says that the consistency of tags is hard to maintain. For instance, a respondent says that "The people that add labels must be consistent with eash other and up-to-date with the current labelling policy."
4) 3 respondents mention that selecting tags from tag library is time-consuming. A respondent mentions that "Volume of issues can be time consuming to tag correctly." Therefore, a tag recommendation approach is required to save developers' time of selecting tags.
5) 2 respondents says that it is difficult to update tags according to pull requests' status changes.
6) 4 respondents mentioning other difficulties. For example, a response is "Need to remember all labels."
7) 8 respondents do not fill in any information about difficulties.

Usefulness of Tag Recommendation. In the fourth question, we ask developers whether it is useful or useless if there is a tool to recommend tags for pull requests, and plot their responses in Table 5. 60.61% of respondents consider a recommendation tool as useful, while 27.27% of respondents consider a recommendation tool as useless. 12.12% of respondents are unsure. The majority of respondents think that a tag recommendation tool is useful.

We take a further step and ask detailed reasons of their choices. More specifically, we ask developers two questions: If the recommendation tool is useful, what features should be considered in recommendation tool? If the recommendation tool is useless, why? 4 respondents explain reasons why the recommendation tool is useless. They think that it is difficult to recommend tags, and the logic to apply labels is too hard to figure out. A recommendation tool is appreciated by 20 respondents. Table 6 shows suggestions for implementing a recommendation tool. From results, we can note that:

1) 5 respondents agree that a recommendation tool should consider pull requests' code. For instance, a respondent writes that "Identifying the components affected by the issue/PR". Another respondent writes that "Have the tool look at what parts of the program were changed and what kind of changes and act accordingly, at least for subsystem specific labels. "
2) 3 respondents believe that text information should be considered in a recommendation tool. For instance, a respondent mentions that 'Go through title/description to suggestion of possible tags". Another respondent suggests that "Keyword detection in pr description".
3) 3 respondents point out that the recommendation tool needs to use the history information. For example, a respondent mentions that "Being able to learn on its own based on corrected labels applied by the project maintainers."
4) 3 respondents agree that this recommendation tool should be automatic. For example, a respondent says that "Should be as automated as possible."
5) 6 respondents mention other features. For example, a respondent says that "It needs to be well-integrated into the GitHub user interface."

Survey results show that tags are used to describe functions, priorities, statuses and components, which helps developers to track, search or classify pull requests. However, some respondents think that it is difficult to choose right tags and keep consistency of tags. Meanwhile, it is time-consuming to select tags from the tag library. In order to solve these problems, we further ask respondents' attitude towards a tag recommendation tool. The majority of respondents think that a tag recommendation tool is useful, and this recommendation tool should consider code, text and history information. According to survey results, we design a tag recommendation method in Section 4.

Table 3
What are benefits of setting tags for the pull requests?.

the benefits of tags	Respondents
Track pull requests	12 / 36.36%
Search pull requests	12 / 36.36%
Classify pull requests	5 / 15.15%
Other	6 / 18.18%

Table 4
What are difficulties in setting tags for pull requests?.

Difficulty	Respondents
Choose appropriate tags	8 / 24.24%
Create new labels	5 / 15.15%
Set tags consistently	4 / 12.12%
Time-consuming	3 / 9.09%
Update tags	2 / 6.06%
Other	4 / 12.12%
No responses	8 / 24.24%

Table 5
Will it be useful or useless if there is a tool to recommend tags for pull requests?.

Choice	Respondents
Useful	20 / 60.61%
Useless	9 / 27.27%
Unsure	4 / 12.12%

Table 6
What features should be considered in recommendation tool?.

Feature	Respondents
Code	5 / 25%
Text	3 / 15%
History	3 / 15%
Automatic	3 / 15%
other	6 / 30%

4. Tag recommendation approach

In this section, we describe our tag recommendation method FNNRec which uses feed-forward neural network to analyze titles, description, file paths and contributors. As shown in Fig. 2, the entire framework contains two phases: a training phase and a recommendation phase. In training phase, our goal is to build a feed-forward neural network from historical information. In recommendation phase, the network is used to recommend tags for pull requests.

In training phase, FNNRec first collects various information from a set of training pull requests with known tags. We extract titles and description (Step 1), file paths (Step 2) and contributors (Step 3) of pull requests from crawled information. We describe detailed definitions and why we choose these elements in Section 4.1, 4.2 and 4.3. Since we recommend tags for pull requests immediately after their submission, we do not consider any information which are generated in code review process, such as reviewers or commenters. According to pull requests' information and their tags, we build feed-forward neural network (Step 4). We do not consider pull requests' creation time, and pull requests are treated as a set without consideration of the order in which they were created.

In recommendation phase, we use FNNRec to predict whether a tag is likely to be assigned to a specific pull request. FNNRec first extracts titles and description (Step 5), file paths (Step 6) and contributors (Step 7). Then, it processes above information into feed-forward neural network built in the training phase (Step 8). This network will output probabilities of tags, and tags with the highest probabilities are recommended (Step 9).

We use the feed-forward neural network in tag recommendation because it is widely used in classification tasks [17]. The feed-forward neural network has an input layer, hidden layers and an output layer. The advantage of feed forward model is that it is a simple form of the neural network, and information is only processed in one direction from the input layer to the output layer, without going backward or entering any loops. The feed-forward neural network can classify nonlinear separable patterns by nonlinear activation functions and approximates an arbitrary continuous function. We compare the feed-forward neural network with other machine learning algorithms or deep learning algorithms in Section 5.7, and results show that feed-forward neural network achieves the best performance.

4.1. Title and description (step 1)

The text information is often used in the developer recommendation for bug resolution [18–20]. When contributors submit pull requests, they write titles and description to briefly introduce code changes they make. As described in Section 3, a respondent mentions that "Go through title/description to suggestion of possible tags". The intuition is that similar pull requests are often described in a similar way, and they may have similar tags. Therefore, we consider titles and description to recommend tags. We extract words from titles and description. More specifically, we make tokenization, remove stop words and stem words, then convert words into lowercase by natural language toolkit NLTK.

4.2. File path (step 2)

According to Table 6, 5 respondents think that a recommendation tool should consider pull requests' code. For instance, a respondent writes that "Identifying the components affected by the issue/PR". File paths may show components of modified code. Pull requests with code in similar file paths may modify the same components, and they may be assigned with the same tags. Therefore, file paths are analyzed to recommend tags for pull requests. Previous works [21,22] also use file paths to measure code's locations for reviewer recommendation. This is because files in similar locations may have related functions, and need code review from the same reviewers.

Following previous work [21], we use the separator '/', and extract words from file paths. We also take the pull request with number 21,481 in Fig. 1 as an example. This pull request has one modified file path, namely "src/common/Preforker.h". For this file path, we extract three words from the path, including "src", "common" and "Preforker.h". Some pull request has several file paths, and we extract words from all modified file paths.

4.3. Contributor (step 3)

In our survey, some respondents think that recommendation tool needs to use the history information, and contributors are important

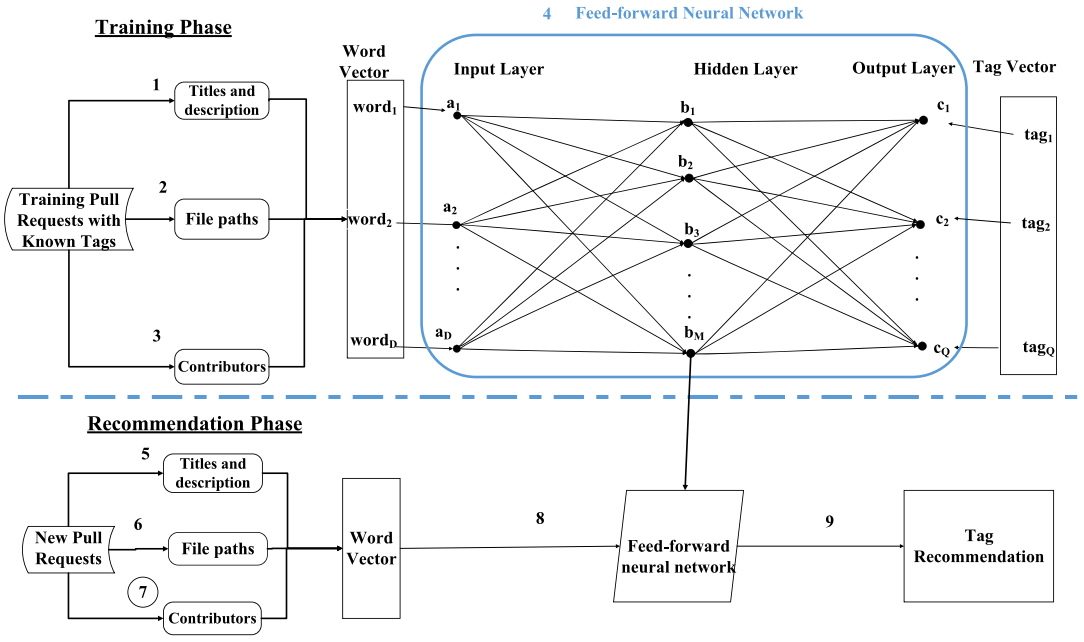


Fig. 2. Overall framework of FNNRec.

history information of pull requests. Some open source projects have much code, and contributors may be familiar with some parts of projects. Furthermore, contributors are not experts in all fields, and they may have some interests in some specific fields. Contributors may submit several similar pull requests, which may be assigned with the same tags. Contributors are extracted as words and used in tag recommendation.

4.4. Feed-forward neural network (step 4)

The next step is to build feed-forward neural network in training phase. We first extract words from pull requests' titles, description, file paths and contributors. We use words of all pull requests in training datasets to construct a vocabulary. Then we build a word vector for each pull request. The length of word vector is the number of words in the vocabulary. Each element in word vector stands for the number of times that the word appears in a pull request's title, description, file paths and contributor. We remove words which appear less than 5 times in all pull requests, so as to decrease the length of word vector and save the training time.

Next, we build a tag vector for each pull request. The length of tag vector is the number of tags in project's tag library. This tag library includes all tags which are used in training datasets, and excludes new tags which are never used in current training datasets. Each element in tag vector is the probability that the tag is used in the pull request. If this tag is assigned to the pull request, the probability is set as 1; otherwise, the probability is set as 0.

Then we analyze training datasets, and build feed-forward neural network. As shown in Fig. 2, pull requests' word vectors construct input layers, and their tag vectors construct output layers. a_h in Fig. 2 stands for the number of times that the word appears in a pull request's title, description, file paths and contributor. D is the length of word vector, namely the number of words in the vocabulary. c_j in Fig. 2 stands for probability that the tag is used in the pull request. Q is the length of tag vector, namely the number of tags in tag library. Pull requests in training datasets are input to determine the best weights and build feed-forward neural network. The number of hidden units M is set as D by default. We discuss the setting of M in Section 5.6.

According to previous work [23], detailed training steps of feed-forward neural network are described as follows:

(1) Training datasets are used to compute the value of a unit a_h in input layer, namely the number of times that corresponding word appears in a pull request's title, description, file paths and contributor. Then units in input layer are converted to units in hidden layer by activation function conversion [24]. We use b_s to represent value of hidden unit s. The value of hidden unit is calculated as follow:

$$b_s = f_H\left(\sum_{h=1}^{D} w_{hs}a_h + \gamma_s\right) \tag{1}$$

where w_{hs} is the conversion weight from input unit h to hidden unit s, and γ_s is bias of the hidden unit s. f_H is the activation function of hidden layer [24].

(2) Units in hidden layer are converted to units in output layer by another activation function conversion. We use b_s to represent value of hidden unit s, and use c_j to represent predicted value of output unit j, namely the predicted probability that the tag is used in the pull request. The predicted value of output unit is calculated as follow:

$$c_j = f_O\left(\sum_{s=1}^{M} v_{sj}b_s + \theta_j\right) \tag{2}$$

Where v_{sj} is the conversion weight from hidden unit s to output unit j, and θ_j is bias of output unit j. f_O is another activation function of output layer [24].

(3) Output layer generates predicted values of output units. Training datasets are used to determine actual values of output units. For example, if a tag is actually used in a pull request, the actual value of corresponding output unit is 1; otherwise, the actual value of corresponding output unit is 0. The goal of training phase is to reduce errors between predicted values and actual values of units in output layer. We use $Y(Y = (c_1, c_2, \ldots, c_Q))$ to represent the predicted values, and use $\widehat{Y}(\widehat{Y} = (\widehat{c}_1, \widehat{c}_2, \ldots, \widehat{c}_Q))$ to the represent actual values. Then we define loss function of the difference as follows:

$$loss\left(\widehat{Y}, Y\right) = -\sum_{i=1}^{Q}\left[\widehat{c}_i log\frac{e^{c_i}}{1+e^{c_i}} + \left(1-\widehat{c}_i\right)log\frac{1}{1+e^{c_i}}\right] \tag{3}$$

We use N to represent the scale of training datasets. Total loss for training datasets is computed by fitness function as follows:

$$Loss(W, V, \Gamma, \Theta) = \sum_{l=1}^{N} loss\left(\widehat{Y}_l, Y_l\right) \tag{4}$$

W is the conversion weight from input layer to hidden layer, and Γ is the bias matrix. V is the conversion weight from hidden layer to output layer, and Θ is the bias matrix.

(4) The goal of training phase is to reduce errors between predicted values and actual values of units in output layer. In order to achieve this goal, the task is to find the best W, V, Γ and Θ which minimizes fitness function in Eq. (4). The most popular approach to minimize fitness function is the back propagation algorithm. Gradient descent is used to update weights W, V and biases Γ, Θ by propagating errors of output layer successively back to hidden layers. Details are described in previous work [11]. The best W, V, Γ and Θ are used to determine feed-forward neural network, which is used in recommendation phase.

4.5. Recommendation phase (step 5 to step 9)

Given a new pull request, we build its word vector based on its title and description (Step 5), file paths (Step 6) and contributors (Step 7). Then we use feed-forward neural network built in the training phase to compute its tag vector, which describes probabilities that tags are assigned to new pull request (Step 8). Tags with the highest probabilities are recommended to the new pull request (Step 9).

5. Evaluation

In this section, we present results of our evaluation for proposed approach. The aim of this study is to investigate the effectiveness of FNNRec approach in providing tag recommendation solutions. We first present evaluation procedure, research questions and evaluation metrics. We then present our experiment results that answer these research questions.

5.1. Evaluation procedure

In order to simulate the usage of methods in practice, we sort all tagged pull requests in chronological order of their creation time. Since feed-forward neural network [11] needs a certain amount of training datesets, we collect the first 2000 tagged pull requests as the first training set, and the 2,000-th pull request's creation time is set as T_1. Interval time M is used to measure the time length in a testing set. Then we build training sets and testing sets. For the N^{th} round, pull requests created before $(T_1 + (N-1)*M)$ months are used to build a training dataset, and pull requests created between $(T_1 + (N-1)*M)$ months and

($T_1 + N^*M$) months are used to build a testing dataset. Interval time M is set as 1 by default, and we discuss the setting of M in Section 5.6. For example in the first round, the training set is built by pull requests created created before T_1, and the testing set is built by pull requests created between T_1 and $T_1 + 1$ month. We use the similar way to build other training sets and testing sets. We use the training set and the testing set to compute the performance of FNNRec in each round, and then compute average values of tagged pull requests. This setup ensures that only past pull requests are used to make the recommendation, and all pull requests in a training set are created before pull requests in a testing dataset. In each round, we build a training dataset and a testing dataset. Table 7 shows the number of rounds in projects, when we consider the interval time as 1 month. 9 projects have at least 20 rounds.

5.2. Research questions

We are interested to answer following research questions:

RQ1: *How effective is FNNRec in recommending tags? How does FNNRec compare with TagDeepRec [10] and TagMulRec [8]?*

In order to evaluate the efficiency of our approach FNNRec, we compare it with approaches TagDeepRec [10] and TagMulRec [8]. TagDeepRec [10] and TagMulRec [8] are designed to recommend tags in question and answer websites, and their original inputs are the description of questions. In order to recommend tags in GitHub, TagDeepRec [10] and TagMulRec [8] use pull requests' titles and descriptions to replace the questions' description. More specifically, TagDeepRec uses the word2vec model to vectorize pull requests' titles and descriptions and then builds a dictionary with words and their corresponding vectors [10]. Then the corresponding vectors are fed into the attention-based Bi-LSTM model to build the recommendation model. TagMulRec [8] first creates an index for pull requests' titles and descriptions and then constructs target candidate sets that include software objects semantically similar to the given software object. Finally, TagMulRec utilizes multi-classification algorithms to rank tags in the target candidate set. The training and testing process of TagDeepRec [10] and TagMulRec [8] are the same as FNNRec, which is described in Section 5.1.

RQ2: *What are benefits of attribute combination in tag recommendation?*

FNNRec combines titles, description, file paths and contributors to recommend tags for pull requests. We wonder whether all these attributes are necessary in tag recommendations. We compare FNNRec with approaches based on parts of attributes.

RQ3: *What are appropriate parameter settings?*

FNNRec is a tag recommendation method based on feed-forward neural network. The number of epochs describe the number of iterations for weight computation. By default, we set the number of epochs as 40. We would like to investigate precisions, recalls and F1-scores for various values of the number of epochs.

In Fig. 2, units in hidden layer are mainly used to connect units in input layer and output layer. The number of hidden units is set as the number input units by default. We would like to investigate how the number of hidden units affect the performance of our approach.

Table 7
Number of rounds with the interval time as 1 month.

Project	Number of rounds
ceph	25
tgstation	20
elasticsearch	27
core	27
symfony	21
rails	36
angular.js	33
RIOT	22
pandas	20
bitcoin	11

As described in Section 5.1, we collect the first 2000 tagged pull requests as the first training set. We wonder how this setting affects approach performance. Furthermore, we add new pull requests to the training set in each round, which provide dynamic training sets. We wonder whether new pull requests in training set improves the performance of tag recommendation.

In experiments, pull requests created before (T_1+ $(N-1)$*M) months are used to build a training dataset in the N^{th} round, and pull requests created between ($T_1 + (N-1)$*M) months and ($T_1 + N$*M) months are used to build a testing dataset. Interval time M is used to measure the time length in a testing set. When interval time M becomes larger, update frequencies of additional training data becomes lower. Interval time M is set as 1 by default. We wonder how the setting of interval time affects tag recommendation.

RQ4: *What is the benefit of the feed-forward neural network in tag recommendation?*

FNNRec uses the feed-forward neural network to recommend tags. We would like to investigate whether the feed-forward neural network achieves better performance than some other machine learning algorithms or deep learning algorithms. We recommend tags based on Extra Tree, KNN, Random Forest, RNN and LSTM, respectively. Then we compare the performance of different algorithms in recommending tags.

Extra Tree aggregates the results of multiple de-correlated decision trees collected in a 'forest' to output the classification result. KNN (K-Nearest-Neighbors) categorizes an input by using its k nearest neighbors. Random Forest is a machine-learning algorithm that aggregates the predictions from many decision trees on different subsets of data. RNN (Recurrent Neural Network) is a class of artificial neural networks where connections between units form a directed graph along a sequence. LSTM (Long short-term memory) is capable of learning long term dependencies in data.

5.3. Evaluation metrics

In order to evaluate FNNRec, we use metrics precision, recall and F1-score. These metrics are commonly used in evaluation of tag recommendation [5,8].

For a pull request pr, T_{pr} includes actual tags which are assigned to this pull request. $TL_{pr,K}$ include top K tags which are recommended for pull request pr. we define $Recall@K_{pr}$ as the percentage of actual tags who are actually recommended.

$$Recall@K_{pr} = \frac{|TL_{pr,K} \bigcap T_{pr}|}{|T_{pr}|} \tag{5}$$

PR is testing set of pull requests and $|PR|$ is the number of pull requests in testing set. $Recall@K$ is the average value of recalls of pull requests in the testing dataset:

$$Recall@K = \frac{\sum_{pr \in PR} Recall@K_{pr}}{|PR|} \tag{6}$$

We define $Precision@K_{pr}$ as the percentage of recommended tags which are actually assigned to the pull request. Given a pull request pr, the top K precision $Precision@K_{pr}$ is defined as:

$$Precision@K_{pr} = \frac{|TL_{pr,K} \bigcap T_{pr}|}{|TL_{pr,K}|} \tag{7}$$

$Precision@K$ is the average value of precisions of pull requests in the testing dataset:

$$Precision@K = \frac{\sum_{pr \in PR}^{|PR|} Precision@K_{pr}}{|PR|} \tag{8}$$

$F1 - score@K$ is a summary metric that combines both precision and recall to measure the performance of the recommendation approach. This metric can evaluate if an increase in precision (recall) outweighs a reduction in recall (precision). It is calculated as the harmonic mean of

precision and recall:

$$F1\text{-}score@K = 2 \cdot \frac{Precision@K * Recall@K}{Precision@K + Recall@K} \quad (9)$$

In order to compare two methods, we define the gain to compare how the method 1 outperforms the method 2. As described in initial study [20], recall gain, precision gain and F1-score gain are defined as follows:

$$Gain_{Recall@K} = \frac{(Recall@K(1) - Recall@K(2))}{Recall@K(2)} \quad (10)$$

$$Gain_{Precision@K} = \frac{(Precision@K(1) - Precision@K(2))}{Precision@K(2)} \quad (11)$$

$$Gain_{F1\text{-}score@K} = \frac{(F\text{-}score@K(1) - F1\text{-}score@K(2))}{F1\text{-}score@K(2)} \quad (12)$$

where $Recall@K(1)$, $Precision@K(1)$ and $F1-score@K(1)$ evaluates the performance of method 1, and $Recall@K(2)$, $Precision@K(2)$ and $F1-score@K(2)$ evaluates the performance of method 2. If the gain value is above 0, it means method 1 has better accuracy than method 2, otherwise method 2 has better recommendation results.

Further, we define the following null hypotheses to assess the statistical validity of results. The alternative hypotheses can be easily derived from the respective null hypotheses.

H-1: There is no statistically significant difference between *Recall@K, Precision@K* and $F1-score@K$ values of FNNRec, TagDeepRec and TagMulRec.

We apply ANOVA test to assess whether the performance of all groups (FNNRec, TagDeepRec and TagMulRec) is significantly different, and apply Holm-Bonferroni method to control for type I errors. In order to analyze effect size, we also compute partial eta η^2 is defined as the ratio of variance accounted for by an effect and that effect plus its associated error variance within an ANOVA study. According to previous work [25], we applied One Way ANOVA test to assess statistically significant difference with $\alpha = 0.05$ between approaches in terms of recalls, precisions and F1-scores. Test purpose is to assess whether the distribution of one of samples is stochastically greater than the other.

5.4. RQ1: Approach comparison

In order to answer RQ1, we consult Tables 8 and 9 to show the performance of FNNRec. On average, FNNRec achieves *Precision*@3, *Recall*@3, $F1-score@3$, *Precision*@5, *Recall*@5 and $F1-score@5$ of 0.447, 0.726, 0.514, 0.317, 0.816 and 0.427. As shown in Table 1, the average number of tags per pull request is less than 2 in 6 projects. The average number of tags per pull request is between 2 and 3 in 3 projects. Only project *elasticsearch* has 3.566 tags per pull request. In top-5 recommendation, FNNRec recommends 5 tags, and at most 2 tags are correct in pull requests which have actually 1 or 2 tags. A few numbers of actual tags causes that the recommendation cannot have high precisions.

In order to compare FNNRec with TagDeepRec [10] and TagMulRec [8], we compute precision gains, recall gains and F1-score gains, assess the statistically significant difference between approaches, and describe results in Table 8–11. On average across 10 projects, FNNRec outperforms TagDeepRec by 59.446%, 66.083%, 62.985%, 44.73%, 48.104% and 46.414% in terms of *Precision*@3, *Recall*@3, $F1-score@3$, *Precision*@5, *Recall*@5 and $F1-score@5$, respectively. Furthermore, FNNRec outperforms TagMulRec by 26.903%, 22.185%, 24.953%, 21.672%, 17.793% and 20.65% in terms of *Precision*@3, *Recall*@3, $F1-score@3$, *Precision*@5, *Recall*@5 and $F1-score@5$, respectively. Clearly, FNNRec outperforms TagDeepRec and TagMulRec across precisions, recalls and F1-scores in all projects.

In Table 12, We apply the ANOVA test to assess whether the performance of FNNRec, TagDeepRec and TagMulRec is significantly different, and apply Holm-Bonferroni correction for multiple comparisons. Results show that most of p-values are smaller than 0.05, and the family-wise error rates are controlled at low-level alpha. In order to analyze effect size, we also compute partial eta η^2 which is defined as the ratio of variance accounted for by an effect and that effect plus its associated error variance within an ANOVA study. If partial eta is between 0.01 and 0.06, the effect size is small; If partial eta is between 0.06 and 0.14, the effect size is median; If partial eta is larger than 0.14, the effect size is big. Table 13 shows that 64.167% of cases belong to the big effect size, and 17.5% of cases belong to the median effect size. Furthermore, Tables 10 and 11 shows that FNNRec records positive gains with statistical significance (with p-values < 0.05) in most of cases. Therefore, we find support to reject Hypothesis H-1 in favor of FNNRec.

In Tables 8 and 9, TagMulRec [8] performs better than TagDeepRec [10]. TagDeepRec uses the word2vec model to vectorize pull requests' titles and descriptions. Word2vec model needs large training datasets, but our datasets in Table 1 maybe not enough for the word2vec model. TagMulRec [8] also performs better than TagDeepRec [10] in the site Freecode which provides the smallest dataset in the previous work [10].

We take a further step and see some examples of tag recommendation. First, in a pull request with number 9200 in project *angular.js*[5], the actual tags include "cla: no" and "type: docs". Our approach FNNRec recommends tags "cla: no", "type: docs" and "cla: yes". Though tag "cla: no" and tag "cla: yes" are mutually exclusive, FNNRec cannot identify deep semantic relationships between tags, and recommends contradictory tags. TagDeepRec [10] recommends tags "type: docs", "cla: yes" and "type: bug", and the only correct tag is "type: docs". TagMulRec [8] recommends tags "cla: yes", "type: bug" and "frequency: moderate", and all of these tags are incorrect. Second, in a pull request with number 9419 in project *angular.js*[6], the actual tags include "cla: yes", "needs: review" and "type: bug". Our approach FNNRec recommends tags "cla: yes", "needs: review" and "component: forms", and the incorrect tag is "component: forms". TagDeepRec [10] recommends the same tags as actual tags and achieves the best performance. TagMulRec [8] recommends tags "cla: yes", "cla: no" and "type: docs", and only the tag "cla: yes" is correct.

RQ1: FNNRec achieves statistically significant higher precisions, recalls and F1-scores than TagDeepRec and TagMulRec.

5.5. RQ2: Benefits of attribute combination

In order to answer RQ2, we use feed-forward neural network to separately analyze titles and description, file paths and contributors for tag recommendation. We compare performance based on different attributes, and plot results in Table 14. Results show that tag

[5] https://github.com/angular/angular.js/pull/9200
[6] https://github.com/angular/angular.js/pull/9419

Table 8
Precision@3 and recall@3 and F1-score@3 of Approaches TagDeepRec, TagMulRec and FNNRec.

Project	Precision@3			Recall@3			F1-score@3		
	TagDeepRec	TagMulRec	FNNRec	TagDeepRec	TagMulRec	FNNRec	TagDeepRec	TagMulRec	FNNRec
ceph	0.302	0.442	**0.529**	0.359	0.616	**0.753**	0.316	0.495	**0.597**
tgstation	0.265	0.3	**0.332**	0.567	0.693	**0.754**	0.345	0.404	**0.444**
elasticsearch	0.236	0.328	**0.473**	0.166	0.315	**0.446**	0.185	0.303	**0.434**
core	0.315	0.342	**0.374**	0.639	0.689	**0.726**	0.398	0.431	**0.465**
symfony	0.388	0.385	**0.496**	0.61	0.615	**0.727**	0.447	0.447	**0.558**
rails	0.231	0.279	**0.375**	0.574	0.7	**0.929**	0.322	0.391	**0.522**
angular.js	0.39	0.386	**0.48**	0.648	0.68	**0.725**	0.434	0.443	**0.51**
RIOT	0.457	0.551	**0.653**	0.422	0.516	**0.609**	0.429	0.521	**0.617**
pandas	0.18	0.268	**0.425**	0.339	0.498	**0.74**	0.225	0.333	**0.519**
bitcoin	0.184	0.279	**0.336**	0.472	0.711	**0.851**	0.259	0.392	**0.471**
Average	0.295	0.356	**0.447**	0.48	0.603	**0.726**	0.336	0.416	**0.514**

Table 9
Precision@5 and recall@5 and F1-score@5 of Approaches TagDeepRec, TagMulRec and FNNRec.

Project	Precision@5			Recall@5			F1-score@5		
	TagDeepRec	TagMulRec	FNNRec	TagDeepRec	TagMulRec	FNNRec	TagDeepRec	TagMulRec	FNNRec
ceph	0.237	0.33	**0.373**	0.47	0.756	**0.853**	0.304	0.443	**0.5**
tgstation	0.18	0.211	**0.225**	0.647	0.795	**0.835**	0.273	0.323	**0.343**
elasticsearch	0.208	0.269	**0.365**	0.249	0.415	**0.557**	0.216	0.311	**0.421**
core	0.214	0.234	**0.262**	0.7	0.753	**0.814**	0.311	0.338	**0.376**
symfony	0.273	0.265	**0.362**	0.69	0.679	**0.862**	0.372	0.363	**0.487**
rails	0.186	0.201	**0.232**	0.781	0.839	**0.953**	0.296	0.319	**0.367**
angular.js	0.28	0.288	**0.339**	0.732	0.743	**0.788**	0.362	0.371	**0.421**
RIOT	0.36	0.433	**0.502**	0.544	0.668	**0.77**	0.425	0.514	**0.596**
pandas	0.143	0.197	**0.292**	0.435	0.589	**0.827**	0.207	0.285	**0.417**
bitcoin	0.149	0.187	**0.216**	0.626	0.782	**0.903**	0.236	0.296	**0.343**
Average	0.223	0.261	**0.317**	0.587	0.702	**0.816**	0.3	0.356	**0.427**

Table 10
Gains and Statistical Results for top-3 recommendation (%).

Project	$Gain_{Precision@3}$ %		$Gain_{Recall@3}$ %		$Gain_{F1-score@3}$ %	
	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec
ceph	75.166 ***	19.683 ***	109.749 ***	22.24 ***	88.924 ***	20.606 ***
tgstation	25.283 ***	10.667	32.981 ***	8.802	28.696 ***	9.901
elasticsearch	100.424 ***	44.207 ***	168.675 ***	41.587 ***	134.595 ***	43.234 ***
core	18.73 ***	9.357	13.615 **	5.37	16.834 ***	7.889
symfony	27.835 ***	28.831 ***	19.18 ***	18.211 ***	24.832 ***	24.832 ***
rails	62.338 ***	34.409 ***	61.847 ***	32.714 ***	62.112 ***	33.504 ***
angular.js	23.077 ***	24.352 ***	11.883 ***	6.618	17.512 ***	15.124 ***
RIOT	42.888 ***	18.512 ***	44.313 ***	18.023 ***	43.823 ***	18.426 ***
pandas	136.111 ***	58.582 ***	118.289 ***	48.594 ***	130.667 ***	55.856 ***
bitcoin	82.609 ***	20.43 ***	80.297 ***	19.691 ***	81.853 ***	20.153 ***
Average	59.446	26.903	66.083	22.185	62.985	24.953

***$p < .001$, **$p < .01$, *$p < .05$

recommendation based on titles and description achieves better performance than tag recommendation based on file paths or contributors. Titles and description are the most important attributes in tag recommendation, because titles and description introduce what changes are analyzed. Attribute combination is useful for tag recommendation. Therefore, FNNRec combines titles and description, file paths and contributors to recommend tags for pull requests.

RQ2: The combination of titles and description, file paths and contributors is effective for tag recommendation.

made in pull requests and/or why they are needed [26]. Contributor is the least important attribute, because the same developers may still submit pull requests with different tags.

In Table 14, tag recommendation based on all attributes achieve $F1-score@3$ as 0.514, which is higher than a single attribute. The best precisions, recalls and F1-scores are achieved when all attributes are

5.6. RQ3: Parameter settings

FNNRec is a tag recommendation method based on the feed-forward neural network. The number of epochs describes the number of iterations for weight computation. We increase the number of training epochs from 10 to 70 with an interval of 10, and evaluate the

Table 11
Gains and Statistical Results for top-5 recommendation (%).

Project	$Gain_{Precision@5}$ %		$Gain_{Recall@5}$ %		$Gain_{F1-score@5}$ %	
	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec
ceph	57.384 ***	13.03 ***	81.489 ***	12.831 ***	64.474***	12.867 ***
tgstation	25***	6.635	29.057 ***	5.031 ***	25.641 ***	6.192
elasticsearch	75.481 ***	35.688 ***	123.695 ***	34.217 ***	94.907 ***	35.37 ***
core	22.430 ***	11.966	16.286 ***	8.101	20.900 ***	11.243
symfony	32.601 ***	36.604 ***	24.928 ***	26.951 ***	30.914 ***	34.16 ***
rails	24.731 ***	15.423 ***	22.023 ***	13.588 ***	23.986 ***	15.047 ***
angular.js	21.071 ***	17.708 ***	7.650	6.057	16.298 ***	13.477 ***
RIOT	39.444 ***	15.935 ***	41.544 ***	15.269 ***	40.235 ***	15.953 ***
pandas	104.196 ***	48.223 ***	90.115 ***	40.407 ***	101.449 ***	46.316 ***
bitcoin	44.966 ***	15.508 **	44.249 ***	15.473 ***	45.339 ***	15.878 **
Average	44.73	21.672	48.104	17.793	46.414	20.65

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

Table 12
P-values of variance analysis for FNNRec, TagDeepRec and TagMulRec.

Project	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
ceph	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$
tgstation	<0.05	<0.05	<0.05	<0.05	$<0.05^b$	<0.05
elasticsearch	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$
core	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
symfony	<0.05	<0.05	<0.05	$<0.05^b$	$<0.05^b$	$<0.05^b$
rails	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$
angular.js	<0.05	<0.05	$<0.05^b$	$<0.05^b$	0.34	$<0.05^b$
RIOT	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$
pandas	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$
bitcoin	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$	$<0.05^b$

^{b} Significant (corrected $\alpha < 0.05$) with Holm-Bonferroni correction.

Table 13
Partial eta for top-3 and top-5 recommendation.

Project	Precision@3		Recall@3		F1-score@3		Precision@5		Recall@5		F1-score@5	
	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec	TagDeepRec	TagMulRec
ceph	0.632	0.024	0.711	0.039	0.677	0.03	0.461	0.001	0.645	0.001	0.539	0.001
tgstation	0.1	0.006	0.201	0.001	0.156	0.004	0.141	0	0.203	0.004	0.163	0
elasticsearch	0.347	0.109	0.47	0.092	0.391	0.096	0.343	0.114	0.51	0.095	0.412	0.107
core	0.119	0.019	0.046	0.003	0.128	0.02	0.211	0.019	0.136	0.016	0.356	0.039
symfony	0.078	0.078	0.085	0.062	0.1	0.094	0.115	0.145	0.226	0.249	0.148	0.184
rails	0.674	0.438	0.74	0.435	0.725	0.481	0.39	0.148	0.457	0.142	0.451	0.18
angular.js	0.355	0.339	0.084	0.008	0.307	0.22	0.15	0.074	0.005	0	0.2	0.09
RIOT	0.577	0.216	0.526	0.128	0.567	0.181	0.624	0.257	0.666	0.215	0.665	0.282
pandas	0.803	0.65	0.538	0.284	0.732	0.546	0.855	0.658	0.597	0.323	0.825	0.637
bitcoin	0.929	0.342	0.917	0.354	0.931	0.376	0.85	0.246	0.89	0.448	0.874	0.316

Table 14
Precisions@K, Recalls@K and F1-scores@K (K=3,5) with different attributes.

Attribute	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
Title & description	0.4	0.658	0.462	0.287	0.75	0.388
File path	0.371	0.613	0.429	0.27	0.706	0.366
Contributor	0.29	0.474	0.333	0.216	0.572	0.293
All	**0.447**	**0.726**	**0.514**	**0.317**	**0.816**	**0.427**

performance of FNNRec. We describe precisions, recalls and F1-scores in Table 15, respectively. Results show that FNNRec achieves the best F1-scores with the number of epochs as 40 or 50. Since 40 epochs are enough to achieve the best performance, we set the number of epochs as 40 by default.

Units in hidden layer are mainly used to connect units in input layer and output layer. The number of hidden units can be set as a specific percentage of the number of input units. In order to study impacts of the number of hidden units, we increase the number of hidden units from 50% to 200% of the number of input units with an interval of 50%, and evaluate performance of FNNRec. Table 16 show precisions, recalls and F1-scores with different number of hidden units. Results show that there is little variation among different numbers of hidden units. When the number of hidden units is set as 100% of the number of input units, $F1-score@3$ and $F1-score@5$ are slightly better than those of other numbers of hidden units. Therefore, we set the number of hidden units

Table 15
Precisions@K, Recalls@K and F1-scores@K (K=3,5) of FNNRec with different number of epochs.

Epoch	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
10	0.342	0.579	0.4	0.237	0.648	0.326
20	0.406	0.677	0.472	0.289	0.76	0.392
30	0.436	0.712	0.502	0.31	0.803	0.418
40	0.447	**0.726**	**0.514**	0.317	**0.816**	**0.427**
50	**0.448**	0.724	**0.514**	**0.318**	0.814	**0.427**
60	0.401	0.655	0.462	0.288	0.749	0.389
70	0.399	0.651	0.459	0.286	0.744	0.387

Table 16
Precisions@K, Recalls@K and F1-scores@K (K=3,5) of FNNRec with different number of hidden units.

Hidden Unit	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
50%	0.443	0.721	0.509	0.314	0.81	0.423
100%	0.447	**0.726**	**0.514**	**0.317**	**0.816**	**0.427**
150%	0.447	0.724	0.513	**0.317**	0.814	0.426
200%	**0.448**	0.724	**0.514**	0.316	0.813	0.426

Table 17
Precisions@K, Recalls@K and F1-scores@K (K=3,5) of FNNRec with different number of tagged pull requests in the first training set.

Pull requests	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
1000	0.434	0.722	0.503	0.306	0.81	0.416
2000	0.447	0.726	0.514	0.317	0.816	0.427
3000	**0.452**	**0.732**	**0.519**	**0.321**	**0.822**	**0.432**

as 100% of input units by default.

Since feed-forward neural network [11] needs a certain amount of training datesets, we collect the first 2000 tagged pull requests as the first training set. We want to explore the impact of minimal number of tagged pull requests in the training set. We increase the value from 1000 to 3000 with an interval of 1000, and evaluate performance of FNNRec. Table 17 shows precisions, recalls and F1-scores with different data amount. Results show that as the minimal number of tagged pull requests increases, precisions, recalls and F1-scores all increases. However, it costs longer time for projects to accumulate enough pull requests for the first training set. In practice, project owners consider their requirement and decide minimal number of tagged pull requests.

As described in Section 5.1, we collect the first 2000 tagged pull requests as the first training set, and add new pull requests to the training set in each round, which provide dynamic training sets. We wonder whether the addition of new pull requests in training set improves the performance of tag recommendation. We study the performance of tag recommendation based on the fixed training set, namely the first 2000 tagged pull requests in the first training set. The tag recommendation based on fixed or dynamic training set has the same testing dataset in each round. Table 18 shows the average performance values across all rounds based on different training datasets. The tag recommendation based on fixed training set achieves 0.428 and 0.359 in terms of $F1-score@3$ and $F1-score@5$, which are worse than the tag recommendation based on the dynamic training set. Therefore, the addition of new pull requests to the training set improves tag recommendation.

Interval time M is used to measure the time length in a testing set and pull requests created before the testing set are used as training data. New training data is added for updating the feed-forward neural network every M months. Larger interval time M means lower update frequencies of additional training data. Here, we investigate how the setting of interval time affects tag recommendation. Table 19 shows the performance of tag recommendation with different interval time M. When M is set as 1, results show that FNNRec achieves the best performance. Additional training data is added every 1 month, which may build a better feed-forward neural network. As interval time M increases, the performance of tag recommendation becomes worse. In this paper, interval time M is set as 1 by default. In practice, project owners can choose the suitable setting of interval time M for open source projects.

RQ3: FNNRec achieves the best precisions, recalls and F1-scores when the number of epochs is set as 40 or 50, and the number of hidden units is set as 100% of the number of input units.

5.7. RQ4: Benefits of the feed-forward neural network

FNNRec uses the feed-forward neural network to recommend tags. In this subsection, we investigate the performance of different machine learning algorithms or deep learning algorithms. We use different algorithms to build a recommendation model, including Extra Tree, KNN,

Table 18
Precisions@K, Recalls@K and F1-scores@K (K=3,5) of FNNRec with different training datasets.

Training set	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
Fixed training set	0.369	0.621	0.428	0.263	0.713	0.359
Dynamic training set	**0.447**	**0.726**	**0.514**	**0.317**	**0.816**	**0.427**

Table 19
Precisions@K, Recalls@K and F1-scores@K (K=3,5) of FNNRec with different interval time.

Interval time	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
1	**0.447**	**0.726**	**0.514**	**0.317**	**0.816**	**0.427**
5	0.431	0.701	0.495	0.306	0.791	0.412
10	0.417	0.679	0.48	0.297	0.771	0.402
15	0.403	0.663	0.466	0.287	0.753	0.390

Table 20
Precisions@K, Recalls@K and F1-scores@K (K=3,5) of different algorithms.

Algorithm	Precision@3	Recall@3	F1-score@3	Precision@5	Recall@5	F1-score@5
Extra Tree	0.402	0.659	0.464	0.289	0.752	0.391
KNN	0.317	0.533	0.369	0.228	0.614	0.311
Random Forest	0.401	0.659	0.463	0.288	0.751	0.389
RNN	0.389	0.360	0.366	0.284	0.434	0.336
LSTM	0.301	0.492	0.346	0.228	0.604	0.309
FNN	**0.447**	**0.726**	**0.514**	**0.317**	**0.816**	**0.427**

Random Forest, RNN and LSTM. Table 20 shows precisions, recalls and F1-scores of different machine learning algorithms or deep learning algorithms. We notice that FNNRec based on FNN performs better than approaches based on other algorithms. Therefore, we choose the feed-forward neural network to recommend tags.

RQ4: The tag recommendation achieves the best performance based on feed-forward neural network.

6. Threats to validity

Threats to external validity relate to generalizability of our study. First, our experimental results are limited to 10 popular projects. We find that FNNRec achieves higher *precision, Recall* and $F1-score$ values than TagDeepRec and TagMulRec, which are based on 10 projects in our datasets. We cannot claim that the same results would be achieved in other projects. Our future work will focus on evaluation in other projects to better generalize results of our method. We will conduct broader experiments to validate whether FNNRec performs well in tag recommendation. Second, our empirical findings are based on open-source software projects in GitHub, and it is unknown whether our results can be generalized to other open-source software platforms. In the future, we plan to study a similar set of research questions in other platforms, and compare their results with our findings in GitHub.

Construct validity threats are related to the degree to which the construct being studied is affected by experiment settings. First, we use precision, recall and F1-score, which are also used by previous works to evaluate effectiveness of tag recommendation approaches [5,8]. Therefore, we believe there is little threat to construct validity. Second, we define some factors to quantitatively measure potential features mentioned by respondents. There may be other measures. For example, some respondents think that a recommendation tool should consider pull requests' code. In this work, we mainly analyze file paths of modified code but do not analyze functions or classes in modified code. In future work, we will try more factors to recommend tags for pull requests, such as source code representation generated by AST-based Neural Network [27]. Third, we send the survey to integrators whoever set tags. This selection of integrators may favor developers with a positive review toward tag recommendation, and may not reflect as well those who find it useless. Furthermore, we provide three choices for developers' attitudes, including 'Useful', 'Useless' and 'Unsure'. A choice on a Likert scale is better to get more of the range on attitudes.

Threats to conclusion validity is concerned with issues that affect the ability to draw the correct conclusion. We conduct a survey to understand tags in pull requests. Two authors manually read replies, build categories, and classify responses into corresponding categories. The category 'other' may include unclear responses. For example, a response about the usage of tags is "Labels are used to group PR, so related contributor developer can review his/her related tagged PR." Since this response does not describe groups of pull requests, we do not know specific usages of tags in detail. In future work, we may try to send emails to some respondents and ask for more details about their responses.

7. Related work

Related work to this study could be divided into three main categories, including issue tag, tag recommendation, and reviewer recommendation.

Issue tag. Some researchers studied tags used in issues of GitHub [3,28,29]. Cabot et al. explored the use of labels to categorize issues in GitHub [3]. Their results revealed using labels favored the resolution of issues. Bissyande et al. found that two most common tags in issues were bug and feature [28]. Izquierdo et al. presented a visualization tool to help managers and developers to better understand how issue labels were employed in their open-source software projects [29].

In GitHub, developers write issue reports to identify bugs and document feature requests [28]. Developers submit pull requests when they want to merge code changes into main repositories [2]. In this paper, we mainly study tags in pull requests, rather than issue.

Tag recommendation. Initial studies [5,6,8–10,30–32] designed approaches to recommend tags in software information sites, such as StackOverflow and Freecode. Xia et al. proposed a method called TAGCOMBINE to automatically recommend tags for software objects [5,33]. Wang et al. proposed a tag recommendation method called EnTagRec which was based on historical tag assignments to software objects [6]. Results showed that EnTagRec made better tag recommendation than TAGCOMBINE [5,33]. Zhou et al. proposed a new software object multi-classification method TagMulRec which recommended tags for large-scale evolving software information sites [8]. Liu et al. proposed an automated scalable tag recommendation method FastTagRec using

neural network-based classification [9]. Li et al. designed a new tag recommendation approach TagDeepRec using attention-based Bi-LSTM [10]. Experiment analysis showed that TagMulRec outperformed EnTagRec [6], and TagDeepRec outperformed FastTagRec [10]. Experiment results show that in the recommendation of pull requests' tags, our approach FNNRec achieves higher precisions, recalls and F1-scores than TagDeepRec and TagMulRec.

Previous work [34] proposed a graph-based approach to assign tags for repositories in GitHub. This work recommended tags to annotate repositories, and helped developers to efficiently search repositories. Different from this work, our approach FNNRec recommends tags for pull requests in a project.

Reviewer recommendation. There have been a number of studies on reviewer recommendation for pull requests in GitHub [21,35–37]. Jiang et al. used support vector machines to analyze integrators' previous decisions, and designed an approach CoreDevRec to recommend integrators for pull requests [21]. Yu et al. built comment networks to predict appropriate reviewers of incoming pull requests in GitHub [36,37].

Different from these works, we solve a different problem and design an automatic approach to recommend tags, rather than reviewers.

8. Conclusion

In this paper, we first make a survey on the usage of pull requests in GitHub. Survey results show that tags are useful for developers to track, search or classify pull requests. However, it is difficult to choose right tags and keep consistency of tags. 60.61% of respondents think that a tag recommendation tool is useful. In order to help developers choose tags, we propose an approach FNNRec. Based on titles, description, file paths and contributors, FNNRec uses feed-forward neural networks to compute probabilities and recommends tags. We evaluate effectiveness of FNNRec on 10 projects containing 68,497 pull requests. We compare it to approaches TagDeepRec [10] and TagMulRec [8]. The experimental results show that on average across 10 projects, FNNRec outperforms approaches TagDeepRec [10] and TagMulRec [8] by 62.985% and 24.953% in terms of $F1 - score@3$, respectively. FNNRec achieves better recommendation performance than TagDeepRec and TagMulRec. Therefore, we believe that FNNRec is useful to find appropriate tags and improve tag setting process in GitHub.

CRediT authorship contribution statement

Jing Jiang: Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Qiudi Wu:** Software, Validation, Investigation. **Jin Cao:** Software, Investigation. **Xin Xia:** Methodology, Writing - review & editing. **Li Zhang:** Conceptualization, Writing - review & editing, Funding acquisition.

Declaration of Competing Interest

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

Acknowledgment

This work is supported by the National Key Research and Development Program of China No. 2018AAA0102304, the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2019ZX-05, Fundamental Research Funds for the Central Universities under Grant No. YWF-20-BJ-J-1018 and the National Natural Science Foundation of China under Grant No. 61732019.

References

[1] G. Gousios, A. Zaidman, M.-A. Storey, A. van Deursen, Work practices and challenges in pull-based development: The integrator's perspective. Proc. of ICSE, 2015, pp. 1–11.Florence, Italy
[2] G. Gousios, M.-A. Storey, A. Bacchelli, Work practices and challenges in pull-based development: the contributor's perspective. Proc. of ICSE, 2016, pp. 285–296. Austin, USA
[3] J. Cabot, J.L.C. Izquierdo, V. Cosentino, B. Rolandi, Exploring the use of labels to categorize issues in open-source software projects. Proc. of SANER, 2015, pp. 550–554.
[4] I. Steinmacher, I.S. Wiese, I. Polato, A.P. Chaves, M.A. Gerosa, M. Wessel, B.M. de Souza, The power of bots: Understanding bots in oss projects. Proc. of CSCW, 2018, pp. 1–19.New York, USA
[5] X. Xia, D. Lo, X. Wang, B. Zhou, Tag recommendation in software information sites. Proc. of MSR, 2013, pp. 287–296.
[6] S. Wang, D. Lo, B. Vasilescu, A. Serebrenik, Entagrec: an enhanced tag recommendation system for software information sites. Proc. of ICSME, 2014, pp. 291–300.
[7] S. Wang, D. Lo, B. Vasilescu, A. Serebrenik, Entagrec++: an enhanced tag recommendation system for software information sites, Empiric. Softw. Eng. 23 (2018) 800–832.
[8] P. Zhou, J. Liu, Z. Yang, G. Zhou, Scalable tag recommendation for software information sites. Proc. of SANER, 2017, pp. 272–282.
[9] J. Liu, P. Zhou, Z. Yang, X. Liu, J. Grundy, Fasttagrec: fast tag recommendation for software information sites, Automat. Softw. Eng. 25 (2018) 675–701.
[10] C. Li, L. Xu, M. Yan, J. He, Z. Zhang, Tagdeeprec: tag recommendation for software information sites using attention-based bi-lstm. Proc. of KSEM, 2019, pp. 11–24. Athens, Greece
[11] M.-L. Zhang, Z.-H. Zhou, Multi-label neural networks with applications to functional genomics and text categorization, IEEE Trans. Knowl. Data Eng. 18 (10) (2006) 1338–1351.
[12] J. Tsay, L. Dabbish, J. Herbsleb, Let's talk about it: evaluating contributions through discussion in github. Proc. of FSE, 2014, pp. 144–154.Hong Kong, China
[13] S. Yu, L. Xu, Y. Zhang, J. Wu, Nbsl: a supervised classification model of pull request in github. Proc. of ICC, 2018, pp. 1–6.Kansas City, USA
[14] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in github. Proc. of FSE, 2015.Bergamo, Italy
[15] G. Gousios, M. Pinzger, A. van Deursen, An exploratory study of the pull-based software development model. Proc. of ICSE, 2014, pp. 345–355.Hyderabad, India
[16] J. Cohen, Weighted chi square: an extension of the kappa method, Educ. Psychol. Meas. 32 (1) (1972) 61–74.
[17] Y.A. Llave, T. Hagiwara, T. Sakiyama, Artificial neural network model for prediction of cold spot temperature in retort sterilization of starch-based foods, J. Food Eng. 109 (3) (2012) 553–560.
[18] J. Anvik, L. Hiew, G.C. Murphy, Who should fix this bug?. Proc. the 28th ICSE, 2006, pp. 361–370.Shanghai, China
[19] D. Matter, A. Kuhn, O. Nierstrasz, Assigning bug reports using a vocabulary-based expertise model of developers. Proc. of MSR, Vancouver, Canada, 2009, pp. 131–140.
[20] X. Xia, D. Lo, X. Wang, B. Zhou, Accurate developer recommendation for bug resolution. Proc. of WCRE, Koblenz, Germany, 2013, pp. 72–81.
[21] J. Jiang, J.-H. He, X.-Y. Chen, Coredevrec: automatic core member recommendation for contribution evaluation, J. Comput. Sci. Technol. 30 (5) (2015) 998–1016.
[22] P. Thongtanunam, C. Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida, K. ichi Matsumoto, Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. Proc. of SANER, Montreal, Canada, 2015, pp. 141–150.
[23] Y. Zhang, S. Wang, G. Ji, P. Phillips, Fruit classification using computer vision and feedforward neural network, J. Food Eng. 143 (2014) 167–177.
[24] S.F. Crone, N. Kourentzes, Feature selection for time series prediction–a combined filter and wrapper approach for neural networks, Neurocomputing 73 (10–12) (2010) 1923–1936.
[25] M.B. Zanjani, H. Kagdi, C. Bird, Automatically recommending peer reviewers in modern code review, IEEE Trans. Softw. Eng. 42 (6) (2016) 530–543.
[26] Z. Liu, X. Xia, C. Treude, D. Lo, S. Li, Automatic generation of pull request descriptions. Proc. of ASE, San Diego, USA, 2019, pp. 1–13.
[27] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, X. Liu, A novel neural source code representation ased on abstract syntax tree. Proc. of ICSE, Montreal, Canada, 2019, pp. 783–794.
[28] T.F. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, Y.L. Traon, Got issues? who cares about it? A large scale investigation of issue trackers from github. Proc. of ISSRE, Washington DC, USA, 2013.
[29] J.L.C. Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, J. Cabot, Gila: Github label analyzer. Proc. of SANER, 2015, pp. 479–483.
[30] T. Wang, H. Wang, G. Yin, C.X. Ling, X. Li, P. Zou, Tag recommendation for open source software, Front. Comput. Sci. 8 (1) (2014) 69–82.
[31] J.M. Al-Kofahi, A. Tamrawi, T.T. Nguyen, H.A. Nguyen, T.N. Nguyen, Fuzzy set approach for automatic tagging in evolving software. Proc. of ICSM, 2010, pp. 1–10.
[32] F.M. Belem, J.M. Almeida, M.A. Goncalves, A survey on tag recommendation methods, J. Assoc. Inf. Sci. Technol. 68 (4) (2017) 830–844.
[33] X.-Y. Wang, X. Xia, D. Lo, Tagcombine: recommending tags to contents in software information sites, J. Comput. Sci. Technol. 30 (5) (2015) 1017–1035.

[34] X. Cai, J. Zhu, B. Shen, Y. Chen, Greta: graph-based tag assignment for github repositories. Proc. of COMPSAC, 2016, pp. 63–72.Atlanta, USA
[35] J. Jiang, Y. Yang, J. He, X. Blanc, L. Zhang, Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development, Inf. Softw. Technol. 84 (2017) 48–62.
[36] Y. Yu, H. Wang, G. Yin, T. Wang, Reviewer recommendation for pull-requests in github: what can we learn from code review and bug assignment? Inf. Softw. Technol. 74 (2016) 204–218.
[37] Y. Yu, H. Wang, G. Yin, C. Ling, Reviewer recommender of pull-requests in github. Proc. of ICSME, 2014, pp. 609–612.Victoria, Canada