TagDeepRec: Tag Recommendation for Software Information Sites Using Attention-Based Bi-LSTM

Can Li[1], Ling Xu[1](✉), Meng Yan[2], JianJun He[1], and Zuli Zhang[3]

[1] School of Big Data and Software, Chongqing University, Chongqing, China
{lican96,xuling,hejianjun}@cqu.edu.cn

[2] Zhejiang University, Hangzhou, China
mengy@zju.edu.cn

[3] Chongqing Materials Research Institute Limited Company, Chongqing, China
zuli1973@163.com

Abstract. Software information sites are widely used to help developers to share and communicate their knowledge. Tags in these sites play an important role in facilitating information classification and organization. However, the insufficient understanding of software objects and the lack of relevant knowledge among developers may lead to incorrect tags. Thus, the automatic tag recommendation technique has been proposed. However, tag explosion and tag synonym are two major factors that affect the quality of tag recommendation. Prior studies have found that deep learning techniques are effective for mining software information sites. Inspired by recent deep learning researches, we propose TagDeepRec, a new tag recommendation approach for software information sites using attention-based Bi-LSTM. The attention-based Bi-LSTM model has the advantage of deep potential semantics mining, which can accurately infer tags for new software objects by learning the relationships between historical software objects and their corresponding tags. Given a new software object, TagDeepRec is able to compute the confidence probability of each tag and then recommend top-k tags by ranking the probabilities. We use the dataset from six software information sites with different scales to evaluate our proposed TagDeepRec. The experimental results show that TagDeepRec has achieved better performance compared with the state-of-the-art approaches TagMulRec and FastTagRec in terms of $Recall@k$, $Precision@k$ and $F1-score@k$.

Keywords: Software information site · Tag recommendation · Long short term memory · Attention mechanism · Weighted binary cross entropy loss function

1 Introduction

Software information sites (e.g., StackOverflow, AskUbuntu, AskDifferent and Freecode) provide a platform for software developers around the world to share and communicate their knowledge [2–5]. These sites enable developers to pose questions, answer questions and search answers they encountered. When posing a new question in the sites, developers need to choose one or more tags from tag repository or create new ones. Tags in these sites play an important role, not only as keywords to summarize the posted content, but also to classify and organize the software objects to improve the performance of various operations [5]. High-quality tags are expected to be concise and accurate, which can capture important features of the software objects. Unfortunately, it is not easy for the developers to choose the appropriate tags to describe the software objects. There are two major factors that affect developers' choices of high-quality tags: tag synonym [7] and tag explosion [6,7]. Tag synonym refers to similar questions may be provided different tags because of insufficient understanding of the problem or lack of relevant expertise. For example, in StackOverflow, we notice that the tag "c#" is usually expressed as "csharp" and "c-sharp", ".net" expressed as "dotnet" and ".net-framework". Some developers may confuse "python2.7" with "python3.5". Tag explosion describes the phenomenon that the number of tags dramatically increases along with continuous additions of software objects. For example, up to now, there are more than 46 thousand tags in StackOverflow. With such huge tags, noise is inevitably introduced and software objects become more poorly classified. This phenomenon has a negative impact on the speed and accuracy of developers' queries.

To address the above issues, several studies on tag recommendation techniques have been proposed in recent years. These approaches can be roughly divided into word similarity-based and semantic similarity-based techniques. word similarity techniques such as TagCombine [2] focus on calculating the similarity based on the textual description. However, the performance of these approaches is limited by the semantic gap because a lot of questions with the same tags have rare common words. Semantic similarity-based techniques (e.g., TagRec [1], EnTagRec++ [25] and TagMulRec [4]) consider text semantic information and always perform significantly better than the approaches that using word similarity calculation. Recently, Liu et al. [5] proposed FastTagRec using neural network classification techniques, which have been proven to improve the accuracy than the comparable state-of-the-art tool TagMulRec. In this work, we compare the effectiveness of our approach with TagMulRec and FastTagRec on six software information sites.

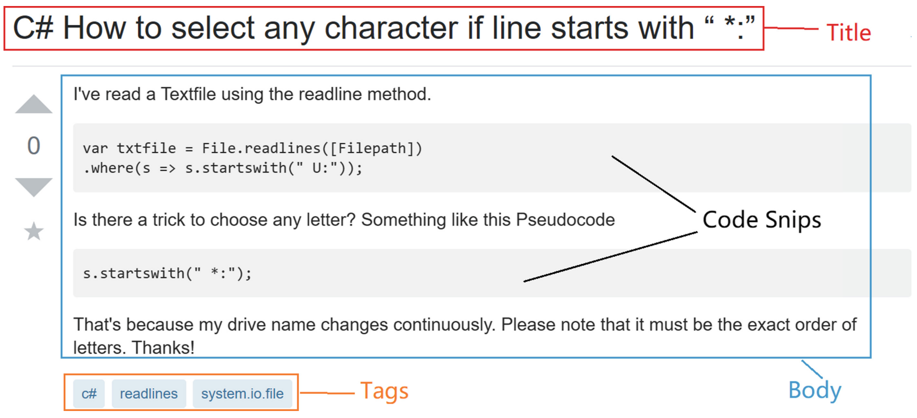
Deep learning has been utilized in other software engineering tasks, such as bug localization [8], code clone detection [9], etc. Inspired by recent Natural Language Processing (NLP) research, in this paper, we proposed a new tag recommendation method called TagDeepRec, a ranking multi-label classification method based on attention-based Bi-LSTM (bidirectional modeling sequence information with long short-term memory). Bi-LSTM has been proven effective in the area of NLP [10–12]. As a deep learning algorithm, Bi-LSTM can capture the deep potential semantic feature by learning the relationships between historical software objects and their corresponding tags. The attention mechanism [13] enables Bi-LSTM model to focus on the key parts of texts associated with tags by attaching more attention weights to the important parts. For a software object, TagDeepRec outputs the confidence probability of each candidate tag by learning from the historical software objects and their corresponding tags, and then k tags with the highest probabilities in the tag candidate set will be recommended to developers as the top-k tags. We evaluate TagDeepRec on six software information sites including StackOverflow@small, Freecode, AskDifferent, AskUbuntu, Unix and StackOverflow@large. StackOverflow@small and StackOverflow@large are divided based on their sizes. Experiments show that TagDeepRec achieves better performance compared with two state-of-the-art approaches: TagMulRec [4] and FastTagRec [5].

Our contributions are as follows:

– We propose an automatic tag recommendation approach for software information sites using attention-based Bi-LSTM model. Experimental results demonstrate that our approach scalable enough to handle both small and very large software information sites.
– We evaluate TagDeepRec on the dataset from six software information sites with different scales. The experiments show that TagDeepRec can achieve high accuracy and outperforms the state-of-the-art approaches TagMulRec and FastTagRec.

The organization of the paper is as follows: In Sect. 2, we give the problem formulation. Section 3 presents our proposed approach. In Sect. 4, we report the results of the experimental evaluation. In Sect. 5, we give the threats to validity of our results. In Sect. 6, we describe the related work. And the conclusion and future work are described in Sect. 7.

2 Problem Formulation


C# How to select any character if line starts with " *:"

I've read a Textfile using the readline method.

```
var txtfile = File.readlines([Filepath])
.where(s => s.startswith(" U:"));
```

Is there a trick to choose any letter? Something like this Pseudocode

```
s.startswith(" *:");
```

That's because my drive name changes continuously. Please note that it must be the exact order of letters. Thanks!

Fig. 1. A software object in StackOverflow

Figure 1 gives a specific example of a software object in StackOverflow. It's observed that this software object consists of a title, a body and three tags. In particular, we can also find code snippets located in the body. Generally, we always treat the combination of a title and a body as a question in a software object.

A software site is a set $S = \{o_1, \cdots, o_n\}$ which is composed of a series of software objects $o_i (1 \leq i \leq n)$. And a software object is also composed of various attributes such as an identifier $o_i.id$, a title $o_i.title$, a body $o_i.body$, several tags $o_i.tags$, etc. We combine the information from title $o_i.title$ and body $o_i.body$ as a new question $o_i.q$. In our method, we assume that each software object contains a question $o_i.q$ and several corresponding tags $o_i.tags$. At the same time, we filter out tags which appear infrequently and define all the remaining tags in a software information site as a candidate tags set $T = \{t_1, \cdots, t_s\}$, where s is the number of remaining tags and $t_i\{1 \leq i \leq n\}$ indicates whether the current tag is selected. The value of t_i is either 0 or 1, where 1 indicates that the current tag is selected and 0 is the opposite phenomenon. Assuming that tags recommended for each software object o_i is a subset of T, given a new software object, the aim of TagDeepRec is to assign several tags from tags candidate set T by learning from existing software objects and their corresponding tags, which can provide great help for developers to choose appropriate tags.

3 Proposed Method

3.1 Approach Overview

Figure 2 presents the overall framework of our TagDeepRec which contains three phases: data preparation, training, and deployment. In the data preparation phase, both the training set and the testing set are derived from the same software information sites. And for a software object, the useful fields including title and body are extracted and further form a question description. After preprocessing, texts of all the question descriptions are collected and formed into a corpus. A word2vec model [15] is used to capture semantic regularities on the corpus. We vectorize all words in the corpus and then build a dictionary with words and their corresponding vectors. In the training phase, each question description represented by text is converted into a vector matrix by word embedding based on the dictionary. Then, the corresponding vectors of question descriptions are fed into the attention-based Bi-LSTM model to detect their features. As the core module of TagDeepRec, the attention-based Bi-LSTM network is a multi-label ranking algorithm, which aims to get the confidence probability of each candidate tag. In the deployment phase, the trained model is loaded to predict tags for the software object.

3.2 Data Preparation

Data Extraction: We first collect historical questions and their tags which are posted on software information sites. For each software object, we only parse out

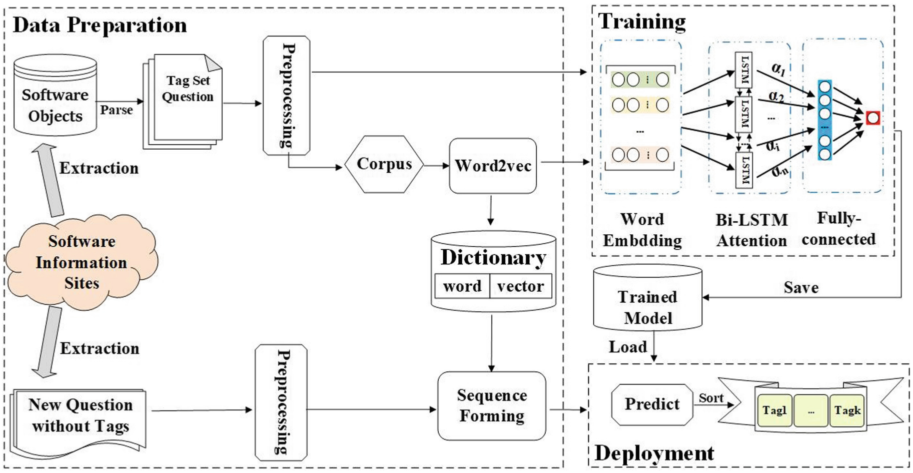


Fig. 2. Overall workflow of TagDeepRec

title, body and tags on the software object and combine the title and the body as a question description. We also remove code snippets in the body, which can easily be located in specific HTML element components ($<code>\cdots<\backslash code>$) [5].

Data Preprocessing: Following previous researches [2–5], we filter out the rare tags whose frequencies of occurrence are less than a predefined threshold θ. And we further remove the software objects if the frequency of all its tags less than θ. These tags rarely appear since few people use them, therefore, they are less useful to be used as representative tags and recommending them provides little help in addressing the tag synonym problem.

For the text information in these software objects, we perform typical preprocessing steps including tokenization, stemming, stop words removal and case conversion by using NLTK (the Natural Language Toolkit) [14]. Stop words removal also deletes some isolated numbers and punctuations.

Matrix Presentation: After preprocessing, all the words in the textual descriptions are made up of the experimental corpus. The word embedding technique (word2vec [15]) with skip-gram model are applied on the corpus to obtain the word vector of each word. Then, we can get a matrix presentation of each question description.

3.3 Attention-Based Bi-LSTM for Tag Recommendation

Given a software object o_i, the matrix presentation of the question $o_i.q = [x_1, \cdots, x_n]$ is fed into attention-based Bi-LSTM model to extract features, where x_i is the vector representation of the $i-th$ word in $o_i.q$. The hidden state $h_i \in \mathbb{R}^d$ denoting the representation of the time step i can be obtained after being processed by the Bi-LSTM layer. Generally, in Bi-LSTM, the hidden state $\overrightarrow{h_i}$ of the forward LSTM is related to its previous memory cell $\overrightarrow{c_{i-1}}$, hidden state $\overrightarrow{h_{i-1}}$ and input vector x_i, while the hidden state $\overleftarrow{h_i}$ of the backward LSTM is related to its next memory cell $\overrightarrow{c_{i+1}}$, hidden state $\overrightarrow{h_{i+1}}$ and input vector x_i, which can be respectively formulated as follows:

$$\overrightarrow{h_i} = f^{(LSTM)}(\overrightarrow{c_{i-1}}, \overrightarrow{h_{i-1}}, x_i) \tag{1}$$

$$\overleftarrow{h_i} = f^{(LSTM)}(\overleftarrow{c_{i+1}}, \overleftarrow{h_{i+1}}, x_i) \tag{2}$$

The final hidden state h_i of time step i $[\overrightarrow{h_i}, \overleftarrow{h_i}]$ is the concatenation of hidden states of both the forward LSTM and the backward LSTM. And we regard H as a matrix consisting of output vectors $[h_1, \cdots, h_n]$, where n is the number of hidden states. H is defined in Eq. (3).

$$H = [h_1, \cdots, h_n] \tag{3}$$

Generally, the tags of a software object are only related to a small portion information of a question. For example, a question like *Binary Data in MySQL, How do I store binary data in MySQL?* Its corresponding tags are {*mysql, binary-data*}. Although there are many words in this question, the words associated with tags are only *mysql, binary* and *data*. In order to capture the key part of software objects, we apply the attention mechanism to pay more attention to the information closely related to tags.

Attention mechanism first attaches weight $\alpha_i\{1 \leqslant i \leqslant n\}$ to the hidden state h_i in our method, which is calculated as follow:

$$M = \tanh(H) \tag{4}$$

$$\alpha_i = Softmax(\omega^T M) \tag{5}$$

where ω is a trained parameter vector and ω^T is a transpose.

Based on the hidden state h_i and its corresponding weight α_i, the context-dependent text representation vector Q of each question is defined as follow:

$$Q = \sum_{i=1}^{n} \alpha_i h_i \tag{6}$$

In the last layer, in order to get a value to represent confidence probability of each tag, we use *Sigmoid* function. Unlike $Softmax$, *Sigmoid* is able to make sure that confidence probability of each tag is independent. Given the input of the fully-connected layer $Q = [q_1, \cdots, q_n]$ and weight vector $W = [w_1, \cdots, w_n]$, the independent probabilities list $TR_i^{predict}$ of candidate tags can be calculated as:

$$TR_i^{predict} = \frac{1}{1 + e^{-(\sum_{i=1}^{n} w_i q_i + b)}} \tag{7}$$

where i is the $i-th$ element of Q, b is the bias vector and n refers to the number of candidate tags.

Considering that the distribution of recommended and not-recommended tags is heavily biased: most of the candidate tags are not be recommended to a software object. Inspired by [16], we set weighted binary cross entropy loss function to balance the loss between positive category and negative category, which is defined as follow:

$$\begin{aligned} L(TR_i^{predict}, TR_i^{actual}) = -\beta TR_i^{actual} \log TR_i^{predict} \\ -(1-\beta)(1-TR_i^{actual}) \log (1-TR_i^{predict}) \end{aligned} \tag{8}$$

where TR_i^{actual} is the actual confidence probabilities list, $TR_i^{predict}$ is the predicted confidence probabilities list, and β is the weight attached to the positive samples.

4 Experiments and Results

In this section, we describe the experiments and results to evaluate the performance of the TagDeepRec.

Table 1. Statistics of six software information sites

Site size	Site name	URL	#Software objects	#Tags
Small	StackOverflow@small	www.stackoverflow.com	47836	437
	AskDifferent	www.apple.stackexchange.com	77503	469
	Freecode	www.github.com	43638	427
Medium	Askubuntu	www.askubuntu.com	246138	1146
	Unix	www.unix.stackexchange.com	103243	770
Large	StackOverflow@large	www.stackoverflow.com	10421906	427

4.1 Experimental Settings

We evaluate the effectiveness of TagDeepRec on six software information sites with different scales. We divide our datasets into three categories: one large-scale site StackOverflow@large, two medium-scale sites AskUbuntu and Unix, three small-scale sites Stackoverflow@small, Freecode and AskDifferent. For comparison, we use the same five datasets as FastTagRec [5] including StackOverflow@large, Askubuntu, Unix, Freecode and AskDifferent. StackOverflow@small is the same dataset used in TagCombine [2], EntagRec [3] and TagMulRec [4]. In detail, for Freecode, AskDifferent, AskUbuntu and Unix, we collects the software objects posted on them before Dec 31st, 2016. StackOverflow@small considers the software objects posted from July 1st, 2008 to December 10th, 2008 and StackOverflow@large selects the software objects posted before July 1st, 2014. We choose the relatively old software objects to make sure our data is stable and reliable. The information of our experimental datasets is shown in Table 1.

In our experiments, we remove the rare tags, which has been described in Sect. 3.2. For the two medium-scale and three small-scale sites, we remove the software object if all of its tags appear less than 50 times and for a large-scale site if all its tags appear less than 10000 times. Table 1 presents the number of *Software Objects* and *Tags* after removing the rare tags.

4.2 Evaluation Metrics

We employ three widely used evaluation metrics [3–5]: top-k recommendation recall (*Recall@k*), top-k recommendation precision (*Precision@k*) and top-k recommendation F1-score ($F1 - score@k$). Given a validation set V composed of n software objects $o_i\{1 \leq i \leq n\}$, we can eventually get the top-k recommended tags $TR_i^{predict}$ and the actual tags TR_i^{actual} of each software object o_i.

Recall@k: *Recall@k* indicates the percentage of tags actually used by the software object coming from the recommended lists $TR_i^{predict}$. For a software object o_i in the validation set V, $Recall@k_i$ of it is computed by Eq. (9), and $Recall@k$, the mean prediction recall of the validation set V is defined by Eq. (10).

$$Recall@k_i = \begin{cases} \frac{|TR_i^{predict} \bigcap TR_i^{actual}|}{|k|}, & |TR_i^{actual}| > |k|; \\ \frac{|TR_i^{predict} \bigcap TR_i^{actual}|}{|TR_i^{actual}|}, & |TR_i^{actual}| < |k|. \end{cases} \tag{9}$$

$$Recall@k = \frac{\sum_{i=1}^{|n|} Recall@k_i}{|n|} \tag{10}$$

Precision@k: *Precision@k* denotes the percentage of the truth tags of a software object in the recommended lists $TR_i^{predict}$. For a software object o_i in the validation set V, $Precision@k_i$ of it is computed by Eq. (11), and $Precision@k$, the mean prediction precision of the validation set V is defined by Eq. (12).

$$Precision@k_i = \begin{cases} \frac{|TR_i^{predict} \bigcup TR_i^{actual}|}{|k|}, & |TR_i^{actual}| > |k|; \\ \frac{|TR_i^{predict} \bigcup TR_i^{actual}|}{|TR_i^{actual}|}, & |TR_i^{actual}| < |k|. \end{cases} \tag{11}$$

$$Precision@k = \frac{\sum_{i=1}^{|n|} Precision@k_i}{|n|} \tag{12}$$

F1-score@k: This metric is the combination of $Precision@k$ and $Recall@k$. For a software object o_i in the validation set V, $F1 - score@k_i$ of it is computed by Eq. (13), and $F1 - score@k$, the mean prediction f1-score of the validation set V is defined by Eq. (14).

$$F1 - score@k_i = 2 * \frac{Precision@k_i - Recall@k_i}{Precision@k_i + Recall@k_i} \tag{13}$$

$$F1 - score@k = \frac{\sum_{i=1}^{|n|} F1 - score@k_i}{|n|} \tag{14}$$

4.3 Experiments Results

Table 2. TagDeepRec vs. FastTagRec & TagMulRec using metrics Recall@k, Precision@k, and F1-score@k

Sites	Recall@5			Precision@5			F1-score@5		
	TagDeep Rec	TagMul Rec	FastTag Rec	TagDeep Rec	TagMul Rec	FastTag Rec	TagDeep Rec	TagMul Rec	FastTag Rec
StackOverflow@small@	**0.817**	0.680	0.805	0.344	0.284	**0.346**	0.463	0.454	**0.482**
AskDifferent	**0.780**	0.708	0.689	**0.405**	0.372	0.357	**0.511**	0.488	0.471
Freecode	0.640	**0.659**	0.588	**0.392**	0.383	0.343	0.449	**0.485**	0.434
Askubuntu	**0.728**	0.603	0.684	**0.361**	0.271	0.346	**0.458**	0.374	0.437
Unix	**0.706**	0.604	0.621	**0.348**	0.294	0.309	**0.447**	0.395	0.397
StackOverFlow@large	**0.885**	0.809	0.870	**0.357**	0.310	0.349	**0.487**	0.449	0.476
Average	**0.760**	0.677	0.709	**0.368**	0.319	0.342	**0.469**	0.441	0.450
Sites	Recall@10			Precision@10			F1-score@10		
	TagDeep Rec	TagMul Rec	FastTag Rec	TagDeep Rec	TagMul Rec	FastTag Rec	TagDeep Rec	TagMul Rec	FastTag Rec
StackOverflow@small	**0.900**	0.777	0.887	**0.194**	0.165	0.187	**0.309**	0.293	0.303
AskDifferent	**0.889**	0.827	0.815	**0.236**	0.222	0.216	**0.361**	0.350	0.342
Freecode	**0.775**	0.758	0.692	0.244	**0.245**	0.219	0.347	**0.364**	0.332
Askubuntu	**0.850**	0.721	0.770	**0.215**	0.166	0.198	**0.331**	0.270	0.303
Unix	**0.835**	0.682	0.722	**0.211**	0.169	0.182	**0.327**	0.271	0.282
StackOverFlow@large	**0.949**	0.892	0.919	**0.195**	0.176	0.187	**0.314**	0.294	0.301
Average	**0.867**	0.776	0.801	**0.216**	0.191	0.198	**0.332**	0.307	0.311

Table 3. Recall@k, Precision@k, and F1-score@k of different TagDeepRec model

	Recall@5	Precision@5	F1-score@5	Recall@10	Precision@10	F1-score@10
Bi-LSTM	0.803	0.336	0.453	0.882	0.187	0.304
Bi-LSTM+Attention	0.812	0.341	0.461	0.892	0.191	0.306
Bi-LSTM+New Loss Function	0.811	0.341	0.460	0.890	0.191	0.305
TagDeepRec	0.817	0.344	0.463	0.900	0.194	0.309

RQ1. Compared with the state-of-the-art approaches, how effective is our TagDeepRec? In order to answer RQ1, we compare TagDeepRec with another two state-of-the-art approaches: TagMulRec [4] and FastTagRec [5] on six software information sites (see Table 1). TagMulRec is an information retrieval tool based on semantics similarity and FastTagRec is a classification technique based on neural network. We evaluate their performances through $Recall@k$, $Precision@k$ and $F1-score@k$ with different k values: 5 and 10. A ten-fold cross validation [24] is performed for evaluation. We randomly split it into ten subsamples. Nine of them are used as training data and one subsample is used for testing. We repeat the process ten times and compute the mean to get more credible results.

The results of these methods are presented in Table 2 and the best values of each site are highlighted in bold. The results show that TagDeepRec achieves a better performance than another two methods except on the Freecode site. Compared to FastTagRec, the average improvement of all sites in $Recall$@5, $Precision$@5, $F1 - score$@5, $Recall$@10, $Precision$@10 and $F1 - score$@10 is 5.1%, 2.6%, 1.9%, 6.6%, 1.8% and 2.1%, and compared to TagMulRec, the average improvement of all sites in $Recall$@5, $Precision$@5, $F1 - score$@5, $Recall$@10, $Precision$@10 and $F1 - score$@10 is 8.3%, 9.0%, 8.4%, 9.1%, 2.5% and 2.5%, respectively. Experimental results also show that the larger the scale of software information site, the better the effectiveness of our approach TagDeepRec.

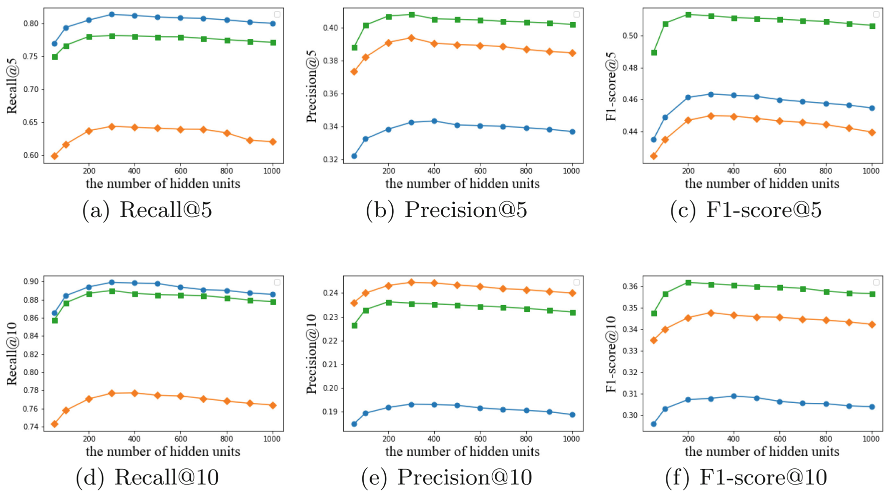
In addition, in order to verify whether TagDeepRec has a statistically significant difference compared with other approaches, we adopt Wilcoxon signed-rank test [17] at 95% significance level on 36 paired value corresponding to the evaluation metrics. Across the six datasets, the corresponding $p - value$ is very small ($p < 0.01$), which confirms that our approach is statistically effective for tag recommendation.

RQ2. Do the attention mechanism and new loss function improve the performance of TagDeepRec? To evaluate the effectiveness of the attention mechanism and new loss function, we designed four different experiments, including the original Bi-LSTM model, attention-based Bi-LSTM model, "Bi-LSTM+new loss function" model and TagDeepRec combined with the attention-based Bi-LSTM model and new loss function. All four experiments are conducted on the StackOverflow@small which is the most commonly used dataset in the field of tag recommendation. And the experimental results are represented in Table 3.

As shown in Table 3, the attention-based Bi-LSTM model outperforms the original Bi-LSTM. The improvement in terms of $Recall$@5, $Precision$@5, $F1 - score$@5, $Recall$@10, $Precision$@10 and $F1 - score$@10 is 0.9%, 0.5%, 0.8%, 1.0%, 0.4% and 0.2% respectively. The result confirms the effectiveness of the attention-based Bi-LSTM model used in our approach. Table 3 also indicates that the enhanced Bi-LSTM model with the new cost function leads to better results as compared to the original Bi-LSTM model. The improvement in terms of $Recall$@5, $Precision$@5, $F1 - score$@5, $Recall$@10, $Precision$@10 and $F1 - score$@10 is 0.8%, 0.5%, 0.7%, 0.8%, 0.4% and 0.1% respectively. The results confirm the usefulness of the new loss function. Additionally, TagDeepRec, which combines the two techniques achieves better performance than another three models.

RQ3.How does the number of hidden states in LSTM affect the performance of TagDeepRec? For RQ3, we choose three small-scale sites including StackOverflow@small, AskDifferent and Freecode to better capture the change of experimental results. We gradually change the values of LSTM hidden states from 50 to 1000 and observe the trend of experimental results. For these three small-scale datasets, Fig. 3(a)–(f) respectively depict the changes of $Recall$@5, $Precision$@5, $F1-score$@5, $Recall$@10, $Precision$@10 and $F1-score$@10 along with the number of hidden states.

The results show that the number of LSTM hidden states affects all the evaluation metrics. For each evaluate metric, there is an optimum range of hidden states. With the number of LSTM hidden states increases, the values of these evaluation metrics rise steadily first, and then gradually decline. Therefore, for small-scale datasets, more hidden states do not always correspond to better performance.



(a) Recall@5 (b) Precision@5 (c) F1-score@5

(d) Recall@10 (e) Precision@10 (f) F1-score@10

Fig. 3. The effect of the number of hidden states in LSTM: StackOverflow@small (blue), AskDifferent (green), Freecode (orange) (Color figure online)

5 Threats to Validity

Internal Validity. Threats to internal validity are the reliability of our experimental data. Because the tags of the software object are chosen freely by developers, errors are inevitable. We assume that the tags of software objects are correct by following prior studies. In order to mitigate this issue, we have taken some measures including selecting common software information sites, selecting the older data in a software information site and filtering out objects with low-frequency and incorrect tags.

External Validity. Threats to external validity are the generalizability of our approach. We have evaluated TagDeepRec on six software information sites and approximately 11 million software objects. In the future, we will further evaluate TagDeepRec from more software information sites to generalize our findings.

Construct validity. Threats to construct validity are the suitability of our evaluation metrics. We use $Recall@k$, $Precision@k$, and $F1-score@k$ to evaluate the proposed TagDeepRec. The metrics used in our study have been widely used in previous studies [3–5]. Thus, we believe we have little threats to construct validity.

6 Related Work

Many approaches have been proposed for automatic tag recommendation recently. TagRec, proposed by Al-Kofahi et al. [1], automatically recommends tags for work items in IBM Jazz. TagRec was based on the fuzzy set theory and took consideration of the dynamic evolution of a system. Xia et al. [2] proposed TagCombine, which is composed of three components including multi-label ranking component, similarity component and tag-term component. TagCombine converts the multi-label classifier model into many binary classifier models. For a large-scale site, TagCombine has to train a large number of binary classifier models. Therefore, TagCombine only work for relatively small datasets. Wang et al. [3] proposed EnTagRec with two components: Bayesian inference component and Frequentist inference component. EnTagRec relies on all information in software information sites to recommend tags for a software object, so it's not scalable as well. Furthermore, Wang et al. [25] proposed an advanced version EnTagRec++ by leveraging the information of users to improve the quality of tags. A more recent approach called TagMulRec was proposed by Zhou et al. [4]. TagMulRec recommends tags by constructing indices for the description documents of software objects and ranking all tags in the candidate set. However, TagMulRec may ignore some important information as it only considers a small part of software information sites which is the most relevant to the given software object. Beomseok et al. [21] proposed a tag recommendation method based on topic modeling approaches, which recommends tags by calculating tag scores based on the document similarity and the historical tag occurrence. Lately, A novel approach called FastTagRec based on neural networks was proposed by Liu et al. [5], which is the prior state-of-the-art work. FastTagRec is not only accurate and scalable, but also faster than existing methods. It recommend tags by learning the relationship between existing postings and their tags.

In the field of software engineering, tags studies have become a hot research problem. Treude et al. [19] implemented an empirical study of a large project on how tagging had been adopted and adapted in the last two years, and showed that tagging mechanism made a contribution to bridging the gap between technical and social aspects of managing work items. Thung et al. [20] concluded that collaborative tagging was useful for detecting similar software applications as a promising information source by performing a user study related to several participants. Wang et al. [21] described the relationships among tags on Freecode and defined them as tag synonyms. Beyer et al. [22] designed TSST, a tag synonym suggestion tool to address the problems of tag synonyms on StackOverflow. And Beyer et al. [23] continued their previous studies and presented an approach to alleviate the issue of tag synonyms by grouping tag synonyms to meaningful topics.

7 Conclusion and Future Work

In this paper, we proposed TagDeepRec, a new tag recommendation approach for software information sites using attention-based Bi-LSTM. We evaluated the performance of TagDeepRec on six software information sites with approximately 11 million software objects. The experimental results show that TagDeepRec is scalable enough to handle both small and large software information sites and has achieved better performance compared with the state-of-the-art approaches (i.e., TagMulRec and FastTagRec). In summary, TagDeepRec can achieve promising performance for the following three reasons: (1) Bi-LSTM model can accurately express the potential semantics of the software objects from both the forward and backward directions. (2) Attention mechanism can capture the key information of the posted questions by attaching different weights to each time step of Bi-LSTM hidden layers. (3) Weighted binary cross entropy loss function can solve the unbalance distribution problems of tags between positive samples and negative samples.

Our current work is based on the question descriptions only. In the future, we plan to consider more features such as code snippets and screenshots to improve effectiveness of tag recommendation. We also intend to explore the tag relationships between the new software object and the historical terms.

References

1. Al-Kofahi, JM., Tamrawi, A., Nguyen, T.T., et al.: Fuzzy set approach for automatic tagging in evolving software. In: 2010 IEEE International Conference on Software Maintenance, pp. 1–10. IEEE (2010)
2. Xia, X., Lo, D., Wang, X., et al.: Tag recommendation in software information sites. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp. 287–296. IEEE (2013)
3. Wang, S., Lo, D., Vasilescu, B., et al.: EnTagRec: an enhanced tag recommendation system for software information sites. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 291–300. IEEE (2014)
4. Zhou, P., Liu, J., Yang, Z., et al.: Scalable tag recommendation for software information sites. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 272–282. IEEE (2017)
5. Liu, J., Zhou, P., Yang, Z., et al.: FastTagRec: fast tag recommendation for software information sites. Autom. Softw. Eng. **25**(4), 675–701 (2018)
6. Joorabchi, A., English, M., Mahdi, A.E.: Automatic mapping of user tags to Wikipedia concepts: the case of a Q&A website-StackOverflow. J. Inf. Sci. **41**(5), 570–583 (2015)
7. Barua, A., Thomas, S.W., Hassan, A.E.: What are developers talking about? An analysis of topics and trends in stack overflow. Empir. Softw. Eng. **19**(3), 619–654 (2014)
8. Deshmukh, J., Podder, S., Sengupta, S., et al.: Towards accurate duplicate bug retrieval using deep learning techniques. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 115–124. IEEE (2017)
9. Li, L., Feng, H., Zhuang, W., et al.: Cclearner: a deep learning-based clone detection approach. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 249–260 IEEE (2017)
10. Cho, K., Van Merriënboer, B., Bahdanau, D., et al.: On the properties of neural machine translation: encoder-decoder approaches. arXiv preprint arXiv:1409.1259 (2014)

11. Zhou, P., Shi, W., Tian, J., et al.: Attention-based bidirectional long short-term memory networks for relation classification. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), vol. 2, pp. 207–212 (2016)
12. Huang, Z., Xu, W., Yu, K.: Bidirectional LSTM-CRF models for sequence tagging. arXiv preprint arXiv:1508.01991 (2015)
13. Vaswani, A., Shazeer, N., Parmar, N., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
14. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python: Analyzing Text With the Natural Language Toolkit. O'Reilly Media Inc., Sebastopol (2009)
15. Mikolov, T., Chen, K., Corrado, G., et al.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
16. Xie, S., Tu, Z.: Holistically-nested edge detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1395–1403 (2015)
17. Wilcoxon, F.: Individual comparisons by ranking methods. Biom. Bull. **1**(6), 80–83 (1945)
18. Hong, B., Kim, Y., Lee, S.H.: An efficient tag recommendation method using topic modeling approaches. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, pp. 56–61. ACM (2017)
19. Treude, C., Storey, M.A.: How tagging helps bridge the gap between social and technical aspects in software development. In: Proceedings of the 31st International Conference on Software Engineering, pp. 12–22. IEEE Computer Society (2009)
20. Thung, F., Lo, D., Jiang, L.: Detecting similar applications with collaborative tagging. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM), pp. 600–603. IEEE (2012)
21. Wang, S., Lo, D., Jiang, L.: Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM), pp. 604–607. IEEE (2012)
22. Beyer, S., Pinzger, M.: Synonym suggestion for tags on stack overflow. In: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, pp. 94–103. IEEE Press (2015)
23. Beyer, S., Pinzger, M.: Grouping android tag synonyms on stack overflow. In: Proceedings of the 13th International Conference on Mining Software Repositories, pp. 430–440. ACM (2016)
24. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. IJCAI **14**(2), 1137–1145 (1995)
25. Wang, S., Lo, D., Vasilescu, B., et al.: EnTagRec++: an enhanced tag recommendation system for software information sites. Empir. Softw. Eng. **23**(2), 800–832 (2018)