The Journal of Systems & Software 170 (2020) 110783

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

TagDC: A tag recommendation method for software information sites with a combination of deep learning and collaborative filtering

Can Li, Ling Xu *, Meng Yan, Yan Lei

School of Big Data & Software Engineering, Chongqing University, Chongqing 400044, China

ARTICLE INFO

Article history:
Received 18 December 2019
Received in revised form 24 July 2020
Accepted 10 August 2020
Available online 16 August 2020

Keywords:
Software information site
Tag recommendation
Deep learning
Collaborative filtering

ABSTRACT

Software information sites (e.g., StackOverflow, Freecode, etc.) are increasingly essential for software developers to share knowledge, communicate new techniques, and collaborate. With the rapid growth of software objects, tags are widely applied to aid developers' various operations on software information sites. Since tags are freely and optionally selected by developers, the differences in background, expression habits, and understanding of software objects among developers may cause inconsistent or inappropriate tags. To alleviate the problems of tag synonyms and tag explosion, we propose TagDC, i.e., a composite Tag recommendation method with Deep learning and Collaborative filtering.

TagDC consists of two complementary modules: the word learning enhanced CNN capsule module (TagDC-DL) and the collaborative filtering module (TagDC-CF). It can improve the understanding of software objects from different perspectives. Given a new software object, TagDC can calculate a list of the combined confidence probabilities of tags and then recommend TOP-K tags by ranking the probabilities in the list. We evaluated our TagDC on nine datasets with different scales. The experimental results show that TagDC achieves a better effectiveness against two state-of-the-art baseline methods (i.e., TagCNN and FastTagRec) with a substantial improvement.

1. Introduction

Software information sites (Wang et al., 2015; Hong et al., 2017; Zhou et al., 2017; Liu et al., 2018) such as StackOverflow, AskDifferent, and AskUbuntu are increasingly essential for software developers. These sites support developers to post online to share knowledge, communicate new techniques, and collaborate (Hong et al., 2017; Zhou et al., 2017; Liu et al., 2018). Generally, the contents (e.g., questions, answers, project descriptions, tags, etc.) of these sites are termed as software objects (Wang et al., 2015; Zhou et al., 2017; Liu et al., 2018). The tag is a brief label consisting of a few words provided as metadata to software objects. Tags are useful for efficiently organizing and classifying the contents of these sites and improving the ease of various operations for developers (Liu et al., 2018). High-quality tags are expected to be concise and accurate enough to summarize the key topics of software objects.

Software information sites allow users to tag their posts in their own words arbitrarily. Recent researches show that it is not easy for them to select consistent and appropriate tags (Zhou et al., 2017; Liu et al., 2018). The choice of tags depends not only on the developers' expertise and language skills, but also on their expression preferences. As a result, the two major challenges for selecting tags are tag synonyms (Joorabchi et al., 2015) and tag explosion (Barua et al., 2014) as the evolution of software information sites.

Tag synonyms describe the phenomenon that tags have the same or similar meanings. Such synonymous tags often have different expressions such as acronyms or full spelling, hyphens or no hyphens, spaces or no spaces, upper or lower cases, etc. (Zhou et al., 2017). Even the widely recognized tags are described differently. For instance, some developers often describe "c#" as "csharp", "javascript" as "js". Tag explosion indicates that the ever-increasing software objects also lead to dramatic growth of tags. By manually investigating the number of tags in each software object, we found that there exist at least one but no more than five tags per posting in the Q&A developer software information sites (e.g., StackOverflow, AskDifferent, etc.) and no more than ten tags per posing in Freecode. As a result, there are more than 58 thousand tags on StackOverflow, more than 5.5 thousand tags on AskUbuntu, and more than 1.1 thousand tags on AskDifferent before July 2019. Liu et al. (2018) found that the objects become increasingly poorly classified with such a vast amount of tags accumulated on these sites, which seriously influences the accuracy and speed of users' queries.

* Corresponding author.
E-mail addresses: 20151611@cqu.edu.cn (C. Li), xuling@cqu.edu.cn (L. Xu), mengy@cqu.edu.cn (M. Yan), yanlei@cqu.edu.cn (Y. Lei).

https://doi.org/10.1016/j.jss.2020.110783
0164-1212/

Table 1
Example of three similar software objects.

NO. 94,058 software object	
Title	Moving from VSS to SVN
Body	I need to write a script to make a source safe project ready to be moved to subversion ····· Is there anything else you can think of that needs to be done before I move the project to SVN?
Tags	***svn***, version-control, ***visual-sourcesafe***
NO. 102,230 software object	
Title	Synchronize SourceSafe with SVN.
Body	Our company has a policy imposing the requirement of keeping source code in a SourceSafe repository. ····· I've tried hard to persuade the management to migrate to SVN with no success.
Tags	***svn***, version-control, synchronization, ***visual-sourcesafe***
NO. 320,694 software object	
Title	Do modern-day VisualStudio .NET projects still use Visual SourceSafe, or SVN?
Body	I'm getting back into .NET after numerous years in PHP/MySQL ····· has SVN become a standard in .NET projects these days?
Tags	.net, ***svn***, ***visual-sourcesafe***

Automatic tag recommendation techniques can deal with these challenges by reusing the existing high-quality tags. Existing automatic tag recommendation techniques can be roughly categorized into two main categories: content-based methods and collaborative filtering based methods. The content-based methods (such as FastTagRec (Liu et al., 2018) and TagCNN Zhou et al., 2019) employ traditional machine learning or deep learning techniques to construct a multi-label classifier based on the contents of historical software objects. These methods can assign several tags by learning from the relationship between software objects and tags. The collaborative filtering based methods (such as TagMulRec Zhou et al., 2017) focus on the tags of similar historical software objects by locating the TOP-N similar objects based on the semantic similarities between the current recommended software objects and historical objects.

In this paper, to combine the advantages of the above two categories, we propose TagDC, i.e., a composite Tag recommendation method with Deep Learning (TagDC-DL) and Collaborative Filtering (TagDC-CF). In detail, TagDC combines the word representation learning enhanced CNN capsule module and the collaborative filtering module.

TagDC-DL is a content-based method, which builds a multi-label classifier through deep learning techniques for tag recommendation. To improve the accuracy of tag recommendation, we use a Bi-LSTM model to extract each word's contextual information to enhance long-distance semantics expression. A followed CNN model with multiple kernels is applied to further extract the local features. Specifically, we leverage the capsule network to output the multi-label confidence probability for each tag due to its superiority in multi-label text classification (Peng et al., 2019; Zhao et al., 2018). Additionally, as shown in Table 1, we find that these three software objects expressing similar topics have the same tags "svn" and "visual-sourcesafe". Hence, we can consider recommending tags for new software objects by reusing the tags attached to similar objects.

TagDC-CF focuses on the similarities among software objects. It is a collaborative filtering technique that can output the similarity-based confidence probability for each tag. We treat these two modules as complementary. Combining deep learning techniques and collaborative filtering techniques is expected to improve our model's effectiveness for more accurate classification.

Then, a linear combination outputs the combined confidence probability corresponding to each tag for new software objects. The TOP-K tags in the candidate tag set with the highest probabilities would be recommended to developers. In our method, the tags recommended to new software objects are selected from our candidate tag set, which is composed of relatively commonly used tags in each site. Such a recommendation mechanism can alleviate the problem of tag explosion and tag synonyms by reducing the creation of inappropriate tags and different tags with similar meanings.

To investigate the effectiveness of our proposed method, we compare the TagDC against two state-of-the-art baselines FastTagRec (Liu et al., 2018) and TagCNN (Zhou et al., 2019) on nine datasets with different scales. These datasets are divided into various scales according to the number of software objects. Considering the weighted average on *Recall*@5 and *Recall*@10 values, TagDC achieves an improvement rate of 18.5% and 23.2% against TagCNN, and an improvement rate of 23.8% and 33.2% against FastTagRec. This demonstrates that our method outperforms two state-of-the-art methods (i.e., FastTagRec and TagCNN) with a substantial improvement. Our study investigates the following research questions:

RQ1: Compared with the state-of-the-art approaches, how effective is our proposed TagDC? **Motivation:** The first research question is performed to evaluate whether the proposed method TagDC outperforms two state-of-the-art tag recommendation methods FastTagRec (Zhou et al., 2019) and TagCNN (Liu et al., 2018). If TagDC shows advantages over two baselines, then our

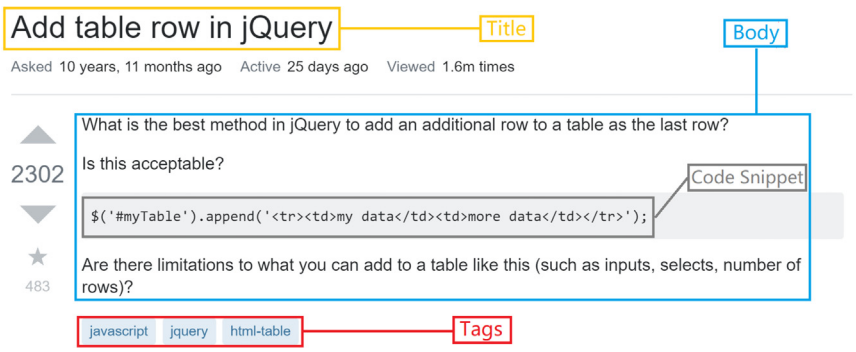


Fig. 1. A software object posted on StackOverflow.

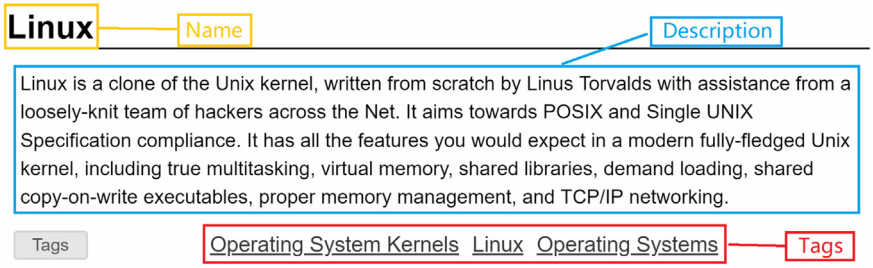


Fig. 2. A project posted on Freecode.

method is beneficial for tag recommendation tasks. **Result:** TagDC outperforms two state-of-the-art methods on all evaluation metrics with a substantial improvement.

RQ2: What is the contribution of TagDC-DL and TagDC-CF to TagDC? **Motivation:** TagDC contains two important modules that affect the model effectiveness. One is TagDC-DL, as described in Section 2.4.1. The other is TagDC-CF, as shown in Section 2.4.2. In this research question, we conduct experiments to evaluate the contribution of TagDC-DL and TagDC-CF to TagDC. **Result:** TagDC combines the advantages of deep learning and collaborative filtering techniques, while TagDC-DL affects the effectiveness most.

RQ3: How effective are different variations of TagDC-DL? **Motivation:** TagDC-DL contains several components (e.g., Bi-LSTM, CNN, capsule networks). This research question aims to evaluate whether each component in TagDC-DL is helpful for experimental results. **Result:** TagDC-DL can benefit from the word representation learning process and capsule networks for tag recommendation tasks.

RQ4: How efficient is TagDC? **Motivation:** Tag recommendation methods are expected to recommend tags for a new posting in negligible time. As shown in Fig. 3, Our TagDC needs to be trained before used for tag recommendation. Reporting the time usage of model training and prediction helps us to better measure the efficiency of TagDC. **Result:** TagDC's time usage for the training and prediction phases is acceptable for various-scale datasets.

The main contributions of this paper are described as follows:

- We propose TagDC, a composite tag recommendation approach combined with two complementary modules: the deep learning module using word learning enhanced CNN capsule (TagDC-DL) and the collaborative filtering module (TagDC-CF). It analyzes and addresses the tag recommendation problem from two different views, and takes advantage of both the semantic similarities among software objects and objects' deep semantic features for more accurate tag recommendation.
- TagDC is evaluated on nine datasets, which are divided into various scales based on the number of software objects. The experimental results show that TagDC outperforms two state-of-the-art baseline approaches (i.e., TagCNN and FastTagRec) with a substantial improvement. The training and prediction time of TagDC are acceptable. It indicates that our method is scalable enough to be applied to various-scale software information sites.

The remaining structure of the paper is as follows: In Section 2, we describe our proposed method in detail. In Section 3, we present the experimental details, including experimental setting, baseline methods, and evaluation metrics, followed by the analysis of experiments results in Section 4. In Section 5, we make a discussion about our method. In Section 6, we review the related work. Finally, we conclude this paper and discuss plans for future work in Section 7.

2. Approach

In this section, we formally formulate our research question and present the overview and the details of our proposed method.

2.1. Problem formulation

Generally, software information sites can be divided into the developer Q&A sites and the developer open-source sites (Liu et al., 2018). The software object in a developer Q&A site such as StackOverflow is a question with several answers. The software object in an open-source developer site such as Freecode is a project with descriptions. Fig. 1 gives a specific example of a software object on StackOverflow. These software objects consist of title, body, and tags. Some postings also include code snippets located in the body. Fig. 2 depicts an open-source project on Freecode that contains the project name, project description, and tags.

A software information site can be regarded as a set $S = \{o_1, \ldots, o_n\}$ that consists of a set of software objects $o_i (1 \leq$

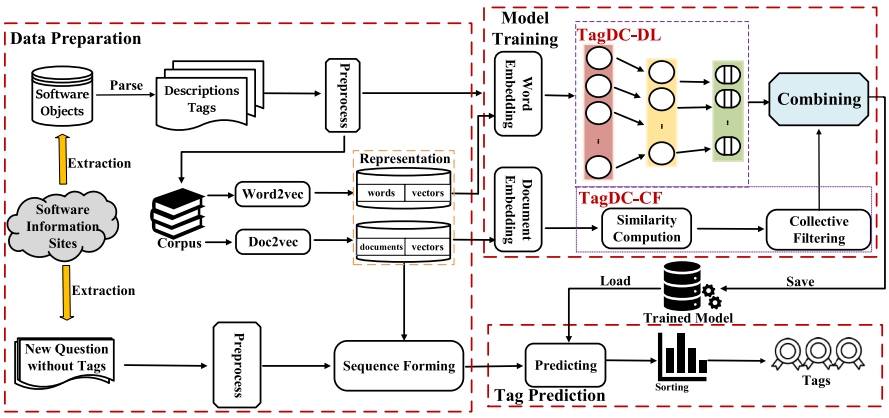


Fig. 3. The overall workflow of TagDC.

$i \leq n$). For the ith software object in a developer Q&A site, the attributes of o_i contain a title $o_i.title$, a body $o_i.body$, several tags $o_i.tags$, etc. Similarly, the attributes of o_i in the developer open-source site contain a description $o_i.d$, several tags $o_i.tags$, etc. We combine the information from $o_i.title$ and $o_i.body$ into a new description $o_i.d$ for the developer Q&A site. Therefore, we can assume that all the software objects contain a description $o_i.d$ and several corresponding tags $o_i.tags$. To alleviate the impact of inappropriate tags, we use the same filtering rule as previous tag recommendation tasks (Zhou et al., 2017; Liu et al., 2018; Zhou et al., 2019). For each dataset, TagDC removes the low-frequency tag with an occurrence frequency of less than or equal to a predefined threshold. The remaining tags are collected to build a candidate tag set. Like one-hot encoding (Rodríguez et al., 2018), we denote the candidate tag set as $T = \{t_1, t_2, \ldots, t_s\}$, where s indicates the number of candidate tags, and the value of t_i defines whether the ith tag is assigned to a software object. The value of t_i is either 0 or 1, where 1 indicates that the ith tag is selected and 0 indicates it is not selected. Assuming that tags recommended for each software object o_i is a subset of T. Given a new software object, TagDC aims to recommend semantically relevant tags to users from tags candidate set T by accurately extracting the features of software objects.

To avoid confusion, we clarify the difference between two terms "software object" and "description". As mentioned above, "Software object" is the basic element of a software information site that contains a set of attributes such as questions, answers, project descriptions, tags, etc. The "description" is an integral part of a software object. There is an inclusive relationship between them. Especially, we only focus on its description and tags for a software object in tag recommendation tasks.

2.2. Overall framework

Fig. 3 presents the overall workflow of our TagDC. The process mainly includes three phases: data preparation, model training, and tag prediction.

In the data preparation phase, TagDC collects historical software objects from the original software information sites. Then, a typical preprocessing process is applied to decrease the noise of descriptions in objects. We adopt both Word2vec (Mikolov et al., 2013) and Doc2vec (Le and Mikolov, 2014) to capture the semantic features from the corpus. For all descriptions in the corpus, Word2vec outputs the corresponding word vectors of all words, while Doc2vec directly outputs their document vectors. Our representation module saves all the word vectors and document vectors of the corpus. In the model training phase, each description would be represented into a matrix for TagDC-DL by querying the vectors of all words through word embedding. Meanwhile, the document vector of each description input to TagDC-CF would be converted through document embedding. Then, TagDC-DL and TagDC-CF output a multi-label confidence probabilities list and a similarity-based confidence probabilities list. A linear combination is then used to weigh the sum of these two lists for the combined confidence probabilities list. Finally, the TOP-K tags with the largest combined confidence probabilities would be recommended in the tag prediction phase.

The following subsections show the details of the three phases.

2.3. Phase I: Data preparation

This subsection shows the details of our data preparation process, including data preprocessing and description representation.

2.3.1. Data preprocessing

We first remove rare tags and software objects. A tag is rare if its number of appearances is less than or equal to a predefined threshold of 50. For a fair comparison, this threshold value is the same as the prior work (Wang et al., 2015; Zhou et al., 2017; Liu et al., 2018; Zhou et al., 2019), and all the experiments are implemented with the same threshold. There are two main reasons for the rare tags mentioned in recent tag recommendation researches (Zhou et al., 2017; Liu et al., 2018; Zhou et al., 2019). One is that the tag is inaccurate. For example, the tag "sql-server" is incorrectly written as "sql-severe" in NO.27674781 software object of StackOverflow. The other one is that tags correspond to relatively rare topics and are not widely recognized by developers (Zhou et al., 2017; Liu et al., 2018; Zhou et al., 2019). Then, we remove a software object if all its tags are rare.

For software objects in a developer Q&A site, we combine a title and a body as a description. We also remove code snippets from a body (see Fig. 1), which is located in the specific HTML element components ($\langle code \rangle \cdots \langle \backslash code \rangle$) (Liu et al., 2018). Similarly, for a developer open-source site, we extract project descriptions and tags from the software objects. In this way, we can ensure that each software object has been extracted as a description with several tags.

To normalize the texts in the descriptions to extract the key features, we use NLTK (natural language toolkit) (Bird et al., 2009) with default parameters for extracting phrases, including case conversion, tokenization, stop words removal, stemming, and special punctuation removal. To remove stop words, we use a standard vocabulary of English stop words. When removing the special punctuation, we keep the special punctuation in some meaningful words such as "c#", "c++", ".net", etc. Finally, by manually investigating the numbers in these descriptions, we

find that tags with numbers account for less than 2% of all tags. Their corresponding descriptions account for less than 1% of all descriptions. Therefore, we can consider that numbers have only a limited effect on the experimental results. Although a small part of the numbers may indicate important information (e.g., version numbers, etc.), they become isolated and meaningless due to the preprocessing process. Thus, we remove them from the text.

2.3.2. Description representation

TagDC uses both Word2vec and Doc2vec for description representation. Among them, Doc2vec has been demonstrated with significant insights for text representation (Lau and Baldwin, 2016). It outputs a global representation for each description for TagDC-CF. However, Doc2vec directly outputs a global document vector for each description, which may miss the rich semantic relations in a description from the word level. In contrast, Word2vec can embed each word in a description into a word vector (Mikolov et al., 2013). The rich semantics features can be extracted from the word level through deep learning techniques. Therefore, for TagDC-DL, we first use Word2vec to embed words into vectors, and then use deep learning techniques (i.e., Bi-LSTM, CNN, capsule networks) to extract rich semantic features for each description.

In detail, For TagDC-DL, a description can be represented into a matrix by querying the vectors of all words though Word2vec. We set words that are not in the corpus to zero vectors. Typically, the input of TagDC-DL is assumed to be a fixed shape, but the length of these descriptions is different. Then, we perform a truncation strategy (Williams and Peng, 1990) to ensure that the lengths of all descriptions are set to a fixed value of l. Parts of more than l words are discarded, and the description containing less than l words would be supplemented by zero vectors. In our experiments, the value of l is set based on the length distribution of descriptions in a software information site. For TagDC-CF, Doc2vec can embed a description into document vector directly.

2.4. Phase II: Model training

In this subsection, we present each step of TagDC-DL and TagDC-CF in detail.

2.4.1. TagDC-DL module

As shown in Fig. 4, TagDC-DL is a word learning enhanced CNN capsule module, which involves three sections:

(1) **Word representation learning**. It enhances the semantics expression of the original word vector by combining the word vector with surrounding contextual information.

(2) **Description representation learning**. We use the kernels in the convolutional layer sliding over each description to extract local features and generate the feature maps.

(3) **Tag probability calculating**. By calculating the length of each tag category based on capsule networks, we can obtain a list of multi-label confidence probabilities for each software object. Then several tags with the highest confidence probabilities are assigned to the current object.

Word Representation Learning. Given a software object o_i, let $o_i.d \in \mathbb{R}^{l\times d}$ denote the input description representation extracted from o_i, where l is the length of the description, and d is the word vector size. Let $x_i \in \mathbb{R}^d$ be the ith d-dimensional word vector in the description $o_i.d$. The matrix presentation of the description $o_i.d = [x_1, \ldots, x_l]$ is first fed into a Bi-LSTM layer to extract features. Consider the sentence extracted from NO.562 software object: "Uploading my first decently sized web app to my shared host provided me with a fresh set of challenges, by which I mean, sleepless nights". Understanding the meaning of the words "challenges" needs to consider their context. Typically, the Bi-LSTM model has been proven effective in extracting the long-distance semantics of sequence information from two directions (Lai et al., 2015). It can accurately summarize the contextual information of each word as a hidden state, which denotes the representation of the time step i processed by the Bi-LSTM layer with n units. Generally, in Bi-LSTM, the forward contextual information $\overrightarrow{h_i} \in \mathbb{R}^n$ generated by the forward LSTM is corresponding to its previous memory cell $\overrightarrow{c_{i-1}}$, hidden state $\overrightarrow{h_{i-1}}$, and current input vector x_i. The backward contextual information $\overleftarrow{h_i} \in \mathbb{R}^n$ generated by the backward LSTM is corresponding to its next memory cell $\overleftarrow{c_{i+1}}$, hidden state $\overleftarrow{h_{i+1}}$ and current input vector x_i, which can be respectively computed as shown below:

$$\begin{aligned}\overrightarrow{h_i} &= f^{(LSTM)}(\overrightarrow{c_{i-1}}, \overrightarrow{h_{i-1}}, x_i),\\ \overleftarrow{h_i} &= f^{(LSTM)}(\overleftarrow{c_{i+1}}, \overleftarrow{h_{i+1}}, x_i).\end{aligned} \tag{1}$$

We combine both the word itself and its corresponding contextual information $h_i \in \mathbb{R}^{2n}$ to catch the accurate and comprehensive word representation. The final vector representation $x'_i \in \mathbb{R}^{d+2n}$ of the ith word can be indicated as follows:

$$h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}] \quad x'_i = [x_i, h_i]. \tag{2}$$

Description Representation Learning. For each description, the matrix representation $X \in \mathbb{R}^{l\times(d+2n)} = [x'_1, \ldots, x'_l]$ is then fed into a convolution layer to extract local features. As kernels in the convolutional layer slide over each description, features at different positions can be detected. A zero-padding strategy (Zhang et al., 2017) is adopted to model the boundary of each description. Given a kernel K_i with a bias term b, a feature map c_i can be emitted as Eq. (3):

$$c_i = f(K_i \circ X + b), \tag{3}$$

where f denotes the ReLU activation function. Furthermore, we use multiple kernels to get various feather maps in our model. All I features are arranged as follows:

$$C = [c_1, c_2, \ldots, c_I] \in \mathbb{R}^{l\times I}. \tag{4}$$

Furthermore, we employ three parallel networks with different kernel window sizes (K_i) of 2, 3, 4, respectively. Then we concatenate all the feature maps from the three branches of the convolutional layer as the final description representation $D \in \mathbb{R}^{l\times 3I}$.

Tag Probability Calculating. Recently, capsule networks with dynamic routing have made promising achievements in the field of textual representation (Peng et al., 2019; Zhao et al., 2018; Kim et al., 2018; Xiao et al., 2018). Capsules contain a set of locally invariant neurons that are effective at recognizing spatial relationships in high-level features and further representing these features in a wider space by converting them into vector-outputs rather than scalar-outputs. Therefore, we use a capsule network model after a convolutional layer.

The primary capsule layer is essentially a convolutional capsule layer, which can summarize the detail of the generated higher-level features. The basic semantics feature maps D generated by the convolutional layer are fed into the primary capsule layer to extract higher-level features. When sliding over D, a series of convolution operations are performed in each convolution kernel K_j to output a series of d-dimension capsules. Similar to the calculation in the convolutional layer, the capsule $p_j \in \mathbb{R}$ in the primary capsule is calculated as:

$$p_j = g(K_j \circ D + b), \tag{5}$$

where g denotes the nonlinear Squash activation function, which can limit the length of vector-outputs between 0 and 1 to represent the probability of each class, and b is the bias term.

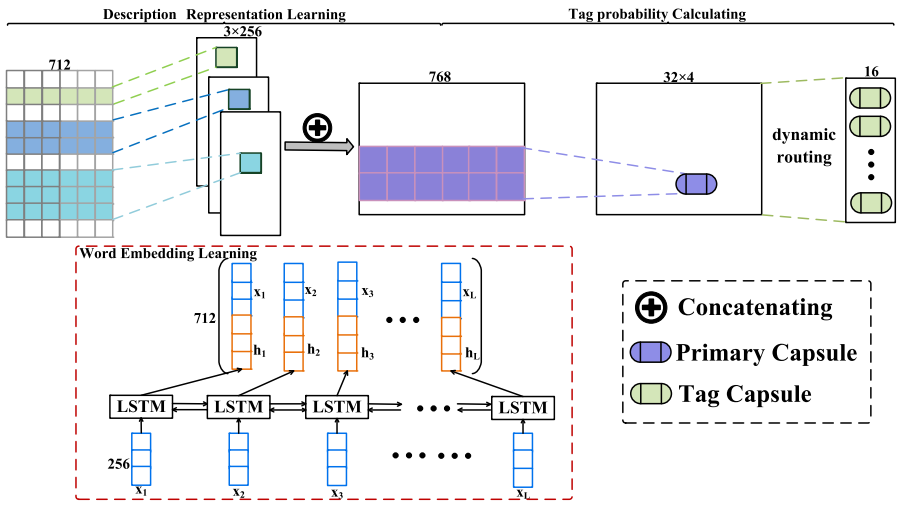


Fig. 4. The Structure of TagDC-DL. Given a description, we first use a Bi-LSTM model to enhance the semantic representation by combining the word vector itself with its context, followed by a convolutional network to extract local features. Finally, a capsule network is used to calculate the confidence probability of each candidate tag for tag recommendation.

With J kernels in the primary capsule layer, the capsule feature maps are assembled as:

$$P = [p_1, p_2, \ldots, p_J]. \tag{6}$$

Then, dynamic routing (Zhao et al., 2018) is performed on the primary capsule layer to generate the capsules for the tag capsule layer, the jth tag capsule v_j is calculated as Eq. (7).

$$\begin{aligned} v_i &= \sum_i c_{ij}\hat{u}_{j|i}, \\ \hat{u}_{j|i} &= W_{ij}p_i, \\ c_{ij} &= \frac{exp(b_{ik})}{\sum_j exp(b_{ik})}. \end{aligned} \tag{7}$$

where $\hat{u}_{j|i}$ is the prediction vector calculated by multiplying the output vector p_i of the primary capsule layer by a weight matrix W_{ij}, and c_{ij} is the coupling coefficient related to the whole iterative process of dynamic routing. Generally, the coupling coefficient indicates the connection strength between two capsules in the adjacent layers. Let b_{ik} be the log prior probability coupled from the ith capsule in the primary capsule layer to the jth capsule in the tag capsule layer. The coupling coefficient c_{ij} is formulated by a *routing softmax* function, which can make sure the sum of all coupling coefficients for the jth capsule is 1.

As shown in Fig. 4, the final tag capsule layer accepts the output-vectors from all capsules in the primary capsule layer. It generates t final tag capsules $v_j, j \in (1, t)$ for classification by dynamic routing mentioned above, where t denotes the number of candidate tag in a software information site. The length of v_j represents the multi-label confidence probability of each tag assigned to a software object. We notice that the length of all tag capsules do not add up to 1. Hence, the capsule network can recognize multiple classes simultaneously, which is suitable for our tag recommendation tasks.

Next, we choose the margin loss objective function defined in Eq. (8) to guide the training process of TagDC-DL.

$$\begin{aligned} L = \sum_{j=1}^{t} &[T_j \max(0, m^+ - \|v_j\|)^2 \\ &+\lambda(1 - T_j)\max(0, \|v_j\| - m^-)^2], \end{aligned} \tag{8}$$

where T_j denotes whether the jth tag exists or not, and the value is 1 if and only if the jth tag is selected by the current description. m^+ and m^- are the thresholds of the upper and lower bounds, which are set to 0.9 and 0.1. $\|v_j\|$ is the length of the tag capsule v_j, λ is a fixed value of 0.5, which can stop the initial learning by decreasing the lengths of all classes' activity vectors (Peng et al., 2019).

Finally, we can obtain a multi-label confidence probabilities list of all tags for each description. The confidence probability list $Tag_i^{TagDC-DL}$ of the ith description is defined as Eq. (9):

$$Tag_i^{TagDC-DL} = [\|v_1\|, \|v_2\|, \ldots, \|v_t\|]. \tag{9}$$

2.4.2. TagDC-CF module

Generally, software objects with similar descriptions correspond to similar tags. Consequently, TagDC-CF is expected to be beneficial to our tag recommendation task. Given a new software object o_i, TagDC-CF first calculates the cosine similarities between it to all historical software objects based on the document vectors of these objects. Then TOP-N software objects most similar to o_i are located. Their tags are used to recommend tags for o_i.

Doc2vec has been demonstrated with significant insights for text representation and achieved higher classification accuracy than other document representation methods in text classification domains (Lau and Baldwin, 2016). Given a software object o_i, Doc2vec can embed its description into a vector D_i.

Let o_j be another historical software object and D_j as its description vector, the similarity between o_i and o_j can be measured by the cosine similarity between D_i and D_j:

$$\delta(D_i, D_j) = \frac{D_i \cdot D_j}{|D_i \parallel D_j|}. \tag{10}$$

Let $T_i = [t_1, t_2, \ldots, t_n]$ be the tags list for o_i, where n is the number of candidate tags. The similarity-based confidence probabilities list calculated in Eq. (11) is the weighted sum over all the similarities of the TOP-N objects most similar to o_i and these objects' tags list T_j.

$$Tag_i^{TagDC-CF} = \sum_{j=1}^{N} \delta(D_i, D_j)T_j. \tag{11}$$

For example, assuming that D_p and D_q are the TOP-2 most similar to D_i with cosine similarity of 0.3 and 0.2 to D_i, and $T_p = [1, 1, 0]$ and $T_q = [1, 0, 1]$ are their corresponding tags

Table 2
Statistics of nine software information sites.

Site size	Site name	Software objects	Tags	Final software objects	Final tags
Small	StackOverflow@small	50,000	9,243	47,836	437
	AskDifferent	77,978	1,049	77,503	469
	Database Administrator	51,031	969	50,687	293
	Freecode	47,978	9,018	43,638	427
	Wordpress	71,338	770	70,491	403
Medium	Askubuntu	248,641	3,041	246,138	1,146
	Serverfault	233,000	3,482	231,319	1,312
	Unix	104,748	2,407	103,243	770
Large	StackOverflow@large	11,203,032	44,265	11,193,763	18,856

list respectively. Based on Eq. (11), the calculation process of the similarity-based confidence probabilities list for D_i is:

$$Tag_i^{TagDC-CF} = 0.3[1, 1, 0] + 0.2[1, 0, 1] = [0.5, 0.3, 0.2].$$

Algorithm 1: How to Adjust the Value of α and β

```
Input: Historical Software Objects Set O
       Tags: Tags for O
       EM: Evaluation Metrics Mentioned in Eq. (15)
       TestSize: The Size of Testing Set
Output: α, β
Initialize The Contribution Weights α = 0, β = 0;
Building TagDC-DL Based on O;
Building TagDC-CF Based on O;
Select a subset As Testing Set T from O with The Size TestSize;
for All Software Object o_i ∈ T do
    Calculate The Multi-Label Confidence Probabilities Tag_i^{TagDC-DL} for o_i;
    Calculate The Similarity-Based Confidence Probabilities Tag_i^{TagDC-CF} for o_i;
end
for α from 0 to 1, Every Time The Increase of α is 0.05 do
    for β from 0 to 1, Every Time The Increase of β is 0.05 do
        Compute Combined Confidence Probabilities Tag_i^{predict} According to
        Eq. (13);
        Evaluate The effectiveness of The Combined Model Based on EM;
    end
end
return α and β Which Produce The Best Result on EM
```

2.5. Phase III: Tag prediction

As shown in the above subsections, given a new software object, TagDC-DL and TagDC-CF respectively output the multi-label confidence probabilities list and the similarity-based confidence probabilities list. Let $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ be the different contribution weights of the multi-label confidence probabilities list and the similarity-based confidence probabilities list, the combined confidence probabilities list for o_i can be formulated by a lineal combination calculation as Eq. (12). The detail of this lineal combination calculation shown in Algorithm 1.

$$Tag_i^{predict} = \alpha Tag_i^{TagDC-DL} + \beta Tag_i^{TagDC-CF}. \quad (12)$$

3. Experimental evaluation

3.1. Experimental settings

For a fair comparison, we evaluate the effectiveness of TagDC by conducting experiments on the same datasets used by our baselines (Liu et al., 2018; Zhou et al., 2019). We also use the same division rule of datasets as our baselines (Liu et al., 2018; Zhou et al., 2019). We define a dataset as a small-scale dataset if the number of its software objects is less than 100,000, as a medium-scale dataset if the number of its software objects is between 100,000 and 1,000,000, and as a large-scale dataset if the number of its software objects is more than 1,000,000. We divide our datasets into five small-scale datasets **StackOverflow@small**, **AskDifferent**, **Database Administrator**, **Freecode**, **Wordpress**, three medium-scale datasets **AskUbuntu**, **Unix**, **Severfault**, and one large-scale dataset **StackOverflow@large**. Among them, StackOverflow@small contains the software objects posted from *July 1st, 2008* to *December 10th, 2008*, while StackOverflow@large selects the posted before *July 1st, 2014*. For the other seven sites, we collect the posted before *December 31st, 2016*. We use the old data in evaluation because older data is relatively stable, as mentioned in past tag recommendation tasks (Wang et al., 2015; Zhou et al., 2017; Liu et al., 2018; Zhou et al., 2019). For each dataset, we divide it into training sets, validation sets, and test sets based on a ratio of 80%-10%-10% according to the posting time of software objects. The older data are used for training.

As described in Section 2.3.1, we should first remove rare tags and their corresponding software objects. To make the experimental results more comparable, we use the same threshold as the baselines (Liu et al., 2018; Zhou et al., 2019). We remove software objects if all of their tags occur no more than 50 times for all datasets. Table 2 presents the statistics of the ten datasets. Columns 2 and 3 list the number of *Software Objects* and *Tags*. Columns 4 and 5 list the number of *Final Software Objects* and *Final Tags* after removing low-frequency software objects and tags.

Furthermore, we design a human study to determine the ground truth of our experimental data. We invite five developers, including four master students and one doctoral student, to investigate whether historical software objects are tagged correctly. All of these developers are experienced in Java and Python.

Since these invited participants are all experienced in Java and Python, we randomly select a total of 100 postings, including 50 postings about Java and 50 postings about Python from StackOverflow. Among these postings, the maximum number of tags is 5, the minimum number of tags is 1, and the average number of tags is 3. All these five developers are invited to investigate whether these 100 postings are correctly tagged in their own time. The postings assigned to each developer are identical. For each tag in a posting, if more than three people mark it as an appropriate tag, we can think it is suitable for the current posting. We collect and analyze responses from these five developers. The results show that 84% of postings are completely tagged correctly, and most of the remaining postings have only one irrelevant tag. Thus, we can believe that our experimental data is reliable.

The parameter settings significantly affect the effectiveness of tag recommendation tasks. For TagDC-CF, we train the Doc2vec model for 25 epochs with a default learning rate.

TagDC-DL needs fixed-length inputs. To determine the appropriate maximum length of descriptions, we made a statistic about the length of all preprocessed descriptions in each dataset. The length distribution of descriptions in each dataset is shown in Fig. 5. We use the upper quartile to represent the maximum length of the description to ensure that most of the descriptions are complete. The maximum length of descriptions is set to 46, 54, 67, 36, 56, 53, 75, 55, 66 for StackOverflow@small, AskDifferent, Database Administrator, Freecode, Wordpress, AskUbuntu, Severefault, Unix, StackOverflow@large, respectively.

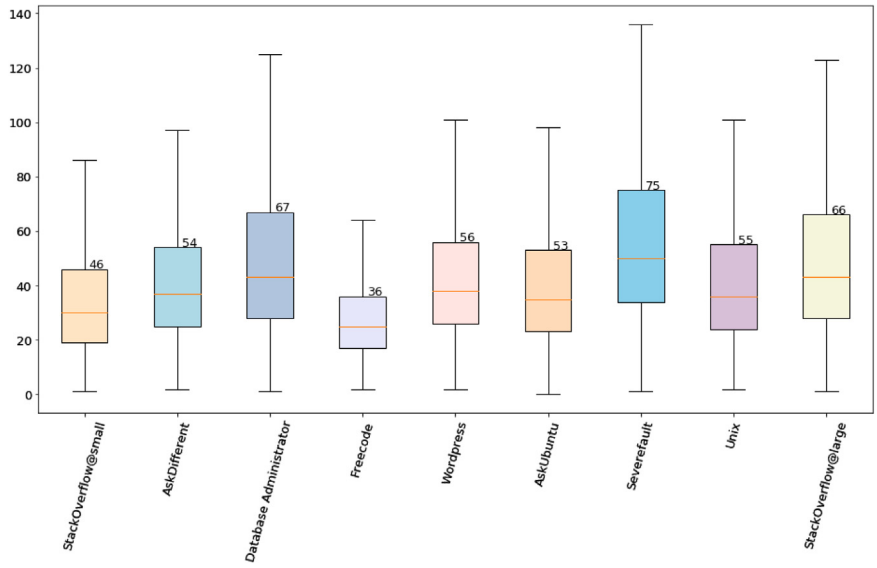


Fig. 5. Length distribution of descriptions in each dataset.

The dimensions of word embedding vectors are empirically set and tuned based on the validation set's effectiveness. For each dataset, we tried the various dimension values from 50 to 500 step by 50, and chose a relatively appropriate dimension value to obtain an acceptable result with an acceptable computational cost. The word embedding vector is set to 200-dimension for five small-scale sites and three medium-scale sites, 300-dimension for the large-scale site. We conduct the size of mini-batch with 400 for StackOverflow@large and 200 for other sites. For the optimization algorithm, we tried Adam, AdaGrad, and RMSProp to train our model, and found that Adam achieved the best effectiveness. Furthermore, Kingma and Ba (2014) found that Adam combines the advantages of AdaGrad and RMSProp. The learning rate also significantly affects the process of model training. If it is too small, too many steps are required for acceptable effectiveness. On the contrary, a large learning rate will likely lead to oscillation. Therefore, we use the Adam optimization algorithm with a learning rate scheduler beginning at 0.001 and shrinking with a 0.1 factor to adjust the learning rate dynamically. An early stopping strategy is used to stop training in time for preventing over-fitting. The network parameters for each layer are also empirically set and tuned based on the validation set's effectiveness. We tried various values for these parameters and selected relatively appropriate ones:

- Bi-LSTM layer: the number of hidden units as 256.
- Convolutional layer: three kinds of kernel size as 2, 3, 4, respectively, the number of kernels as 256, the stride as 1, and the activation function as *ReLU*.
- Primary capsule layer: the dimension of the primary capsule as 32, the kernel size in this layer as 3, the stride as 1, the number of channels as 4, and the activation function as *Squash*.
- Tag capsule layer: the dimension of the tag capsule as 16.

All the experiments are performed on Keras with 8 TITAN V cards and 96G RAM. The operating system and software platforms are Ubuntu 16.04, Python 3.6.4, and Keras 2.1.5.

3.2. Evaluation metrics

To validate the effectiveness of TagDC, we employ three evaluation metrics (*Recall@k*, *Precision@k*, and $F1-score@k$), which have been widely used in past tag recommendation tasks (Hong et al., 2017; Zhou et al., 2017; Liu et al., 2018). The value of k is set to 5 and 10 according to previous researches (Hong et al., 2017; Zhou et al., 2017; Liu et al., 2018). Let T indicate a test set composed by n software objects $o_i\{1 \le i \le n\}$. Using the trained model, we can eventually recommend TOP-K tags $Tag_i^{predict}$ with the highest probability for each object, and the actual tags set is defined as Tag_i^{actual}. The definitions of these three metrics are expressed as follows:

Recall@k is the percentage of its recommend tags selected from the recommended lists $Tag_i^{predict}$, which is computed by Eq. (13), and the mean prediction recall rate for all software objects in T is defined by Eq. (14).

$$Recall@k_i = \begin{cases} \frac{|Tag_i^{predict} \bigcap Tag_i^{actual}|}{|k|}, & |Tag_i^{actual}| > |k|, \\ \frac{|Tag_i^{predict} \bigcap Tag_i^{actual}|}{|Tag_i^{actual}|}, & |Tag_i^{actual}| < |k|. \end{cases} \tag{13}$$

$$Recall@k = \frac{\sum_{i=1}^{|n|} Recall@k_i}{|n|}. \tag{14}$$

Precision@k is the percentage of true tags in the recommended lists $Tag_i^{predict}$, which is computed by Eq. (15). The mean prediction precision rate for all software objects in T is defined by Eq. (16).

$$Precision@k_i = \frac{|Tag_i^{predict} \bigcap Tag_i^{actual}|}{|k|}, \tag{15}$$

$$Precision@k = \frac{\sum_{i=1}^{|n|} Precision@k_i}{|n|}. \tag{16}$$

$F1-score@k$ can be seen as a harmonic average of *Precision@k* and *Recall@k*. For o_i, $F1-score@k_i$ of it is computed by Eq. (17), and $F1-score@k$, the mean prediction F1-score of the test set T is defined by Eq. (18).

$$F1-score@k_i = 2 * \frac{Precision@k_i \times Recall@k_i}{Precision@k_i + Recall@k_i}, \tag{17}$$

$$F1-score@k = \frac{\sum_{i=1}^{|n|} F1-score@k_i}{|n|}. \tag{18}$$

As mentioned in past tag recommendation tasks (Wang et al., 2015, 2018; Zangerle et al., 2011). *Recall@k* is the primary evaluation metric in tag recommendation fields. Since users often use a small number of tags (less than k) for their postings in practical

applications. As defined in Eq. (15), the value of *Precision@k* is usually small due to the large value of *k*. As defined in Eq. (17), the value of $F1-score@k$ is also influenced by the low value of *Precision@k*.

4. Results and analysis

In this section, we conduct experiments to answer the following four research questions.

4.1. RQ1. Compared with the state-of-the-art approaches, how effective is our proposed TagDC?

Motivation: The first research question is performed to evaluate whether the proposed method TagDC outperforms two state-of-the-art tag recommendation methods FastTagRec (Zhou et al., 2019) and TagCNN (Liu et al., 2018). If TagDC shows advantages over two baselines, then our method is beneficial for tag recommendation tasks.

We compare TagDC with two state-of-the-art techniques (i.e., TagCNN (Zhou et al., 2019), FastTagRec Liu et al., 2018) on nine datasets with different scales (see Table 2). FastTagRec (Liu et al., 2018) constructs a suitable tag recommendation framework by using a single hidden layer neural network. It exploits the rank constraint of words and utilizes shared parameters among features to avoid the limitation in large tag output space. TagCNN (Zhou et al., 2019) applies a convolutional network for tag classification. It employs the kernels in the convolutional layer sliding over each description to extract local features from different positions. For comparison, we re-implement their best-performing model on our dataset by using the open-source codes or experiment settings of these two approaches provided by the authors. Three evaluation metrics *Recall@k*, *Precision@k*, and $F1-score@k$ presented above are used for evaluation, *k* denotes the number of recommended tags, and its value is set to 5 and 10 according to previous work (Hong et al., 2017; Zhou et al., 2017; Liu et al., 2018).

Additionally, to make a more robust statistical comparison for each dataset, we consider the experimental results (*Recall@k*, *Precision@k*, $F1-score@k$) for each software object in the test set. We perform a Wilcoxon signed-rank test (Wilcoxon) at a significance level to analyze the statistical difference between our method and baselines. Then, for each dataset, we also considered all software objects in the test set to get the result distribution difference between our method and baselines. We reported the percentage of the samples whose results achieved by TagDC win/equal/lose the results achieved by two baselines.

Table 3 summarizes the experimental results of three methods. The best values of experimental results are in bold, and the win/equal/lose ratios (W/E/L) between TagDC and the baseline are in italics. From an overall perspective, TagDC achieves competitive results against TagCNN and FastTagRec on all datasets for all six evaluation metrics. To get more accurate results, for each dataset, we weighted average the experimental results with the number of software objects in the test set. Experimental results with the number of software objects in the test set. Considering the weighted average of experimental results, TagDC improves TagCNN by 4.8%, 2.7%, 3.5%, 4.8%, 3.7%, 3.2% in terms of *Recall*@5, *Precision*@5, $F1-score$@5, *Recall*@10, *Precision*@10, $F1-score$@10, and FastTagRec by 7.6%, 4.5%, 6.4%, 7.9%, 5.6%, 5.4% in terms of *Recall*@5, *Precision*@5, $F1-score$@5, *Recall*@10, *Precision*@10, $F1-score$@10. Furthermore, the Wilcoxon signed-rank test results show the experimental results between TagDC and two baselines with a significance level ($p < 0.001$). This confirms that the improvement of TagDC against two baselines is statistically significant. The percentage of the results of TagDC that win/equal/lose the results achieved by two baselines also shows that TagDC achieves better or equal effectiveness on most of the software objects.

Furthermore, to determine whether TagDC achieves significant improvements against two state-of-the-art baseline approaches, we calculate the improvement ratio ξ_{EM} for all evaluation metrics(*Recall@k*, *Precision@k*, $F1-score@k$) according to the distance to 1 (Costa et al., 2016) based on Eq. (19).

$$\xi_{EM} = \frac{EM_c - EM_o}{1 - EM_o}, \tag{19}$$

where EM_c is the result of TagDC, and EM_o is the result of baseline methods in terms of all evaluation metrics.

Comparing the weighted average of the experimental results on three methods, our TagDC achieves the improvement ratio of 18.5%, 4.4%, 7.0%, 23.2%, 4.8%, 4.8% for TagCNN and 23.8%, 7.2%, 12.1%, 33.2%, 7.1%, 7.8% for FastTagRec in terms of *Recall*@5, *Precision*@5, $F1-score$@5, *Recall*@10, *Precision*@10 and $F1-score$@10, respectively. Therefore, we can conclude that the proposed method has substantially outperformed two baseline state-of-the-art methods with a noticeable margin on all datasets. This demonstrates TagDC is suitable for various-scale software information sites.

In theory, FastTagRec only applies a simple single hidden layer neural network for tag recommendation. It was limited to architectures with only a few hidden units for feature extraction. TagCNN can extract the local semantic from different positions of descriptions with n-gram filters. However, Filters with fixed sizes may lose the long-distance semantic in texts. Our method addresses the tag recommendation task by combining the advantages of deep learning techniques and collaborative filtering techniques. Furthermore, three strategies are applied to improve the effectiveness of TagDC-DL. A Bi-LSTM model is first used to extract the context of each word for extracting long-distance semantics. A convolutional network is used for consecutive semantics extraction. A capsule network that is effective at recognizing spatial relationships in high-level features is applied to represent features generated by the convolutional layer in the broader space. The semantics extraction can benefit more with the consideration of consecutive semantics and long-distance semantics.

TagDC outperforms two state-of-the-art methods on all evaluation metrics with a substantial improvement.

4.2. RQ2. What is the contribution of TagDC-DL and TagDC-CF to TagDC?

Motivation: TagDC contains two important modules that affect the model effectiveness. One is TagDC-DL, as described in Section 2.4.1. The other is TagDC-CF, as shown in Section 2.4.2. In this research question, we conduct experiments to evaluate the contribution of TagDC-DL and TagDC-CF to TagDC.

Figs. 6(a) and 6(b) show the results of our tests on TagDC-DL and TagDC-CF and the whole TagDC. Compared with TagDC-DL and TagDC-CF, TagDC consistently performs better than the reduced feature component. From the results, we can observe that TagDC-DL makes a major contribution to the effectiveness of the proposed TagDC. TagDC-CF is the complementary part for TagDC-DL, which helps TagDC achieve about 0.8%-3.0% improvements in terms of *Recall*@5 and 0.5%-1.9% improvements in terms of *Recall*@10 over TagDC-DL. Especially, TagDC outperforms TagDC-DL with a relatively noticeable margin on Freecode.

Table 3
TagDC vs. TagCNN & FastTagRec on nine datasets.

Sites	Recall@5			Precision@5			F1-score@5		
	TagDC	TagCNN	FastTagRec	TagDC	TagCNN	FatTagRec	TagDC	TagCNN	FastTagRec
StackOverflow@small	**0.854**	0.812[a]	0.801[a]	**0.356**	0.343[a]	0.338[a]	**0.481**	0.463[a]	0.452[a]
(W/E/L)	–	*13%/77%/10%*	*16%/75%/9%*	–	*13%/77%/10%*	*16%/75%/9%*	–	*13%/77%/10%*	*16%/75%/9%*
AskDifferent	**0.807**	0.773[a]	0.689[a]	**0.425**	0.404[a]	0.357[a]	**0.529**	0.504[a]	0.471[a]
(W/E/L)	–	*15%/77%/8%*	*29%/64%/7%*	–	*15%/77%/8%*	*29%/64%/7%*	–	*15%/77%/8%*	*29%/64%/7%*
Database Administrator	**0.787**	0.736[a]	0.692[a]	**0.384**	0.354[a]	0.332[a]	**0.501**	0.462[a]	0.449[a]
(W/E/L)	–	*16%/74%/10%*	*26%/70%/4%*	–	*16%/74%/10%*	*26%/70%/4%*	–	*16%/74%/10%*	*26%/70%/4%*
Freecode	**0.715**	0.644[a]	0.588[a]	**0.349**	0.306[a]	0.284[a]	**0.445**	0.389[a]	0.364[a]
(W/E/L)	–	*18%/71%/11%*	*27%/62%/11%*	–	*18%/71%/11%*	*27%/62%/11%*	–	*18%/71%/11%*	*27%/62%/11%*
Wordpress	**0.745**	0.707[a]	0.632[a]	**0.342**	0.312[a]	0.278[a]	**0.441**	0.414[a]	0.386[a]
(W/E/L)	–	*14%/79%/7%*	*25%/69%/6%*	–	*14%/79%/7%*	*25%/69%/6%*	–	*14%/79%/7%*	*25%/69%/6%*
Askubuntu	**0.765**	0.724[a]	0.684[a]	**0.385**	0.360[a]	0.346[a]	**0.488**	0.465[a]	0.437[a]
(W/E/L)	–	*26%/66%/8%*	*27%/64%/9%*	–	*26%/66%/8%*	*27%/64%/9%*	–	*26%/66%/8%*	*27%/64%/9%*
Serverfault	**0.757**	0.699[a]	0.666[a]	**0.394**	0.364[a]	0.344[a]	**0.496**	0.457[a]	0.435[a]
(W/E/L)	–	*23%/68%/9%*	*28%/64%/8%*	–	*23%/68%/9%*	*28%/64%/8%*	–	*23%/68%/9%*	*28%/64%/8%*
Unix	**0.771**	0.685[a]	0.627[a]	**0.373**	0.337[a]	0.309[a]	**0.479**	0.430[a]	0.397[a]
(W/E/L)	–	*21%/69%/10%*	*33%/59%/8%*	–	*21%/69%/10%*	*33%/59%/8%*	–	*21%/69%/10%*	*33%/59%/8%*
StackOverflow@large	**0.756**	0.701[a]	0.682[a]	**0.419**	0.392[a]	0.374[a]	**0.537**	0.501[a]	0.472[a]
(W/E/L)	–	*21%/73%/6%*	*30%/64%/6%*	–	*21%/73%/6%*	*30%/64%/6%*	–	*21%/73%/6%*	*30%/64%/6%*
Weighted Average	**0.757**	0.702	0.681	**0.416**	0.389	0.371	**0.533**	0.498	0.469
Sites	Recall@10			Precision@10			F1-score@10		
	TagDC	TagCNN	FastTagRec	TagDC	TagCNN	FastTagRec	TagDC	TagCNN	FastTagRec
StackOverflow@small	**0.917**	0.890	0.887[a]	**0.198**	0.191	0.187[a]	**0.319**	0.305	0.303[a]
(W/E/L)	–	*10%/82%/8%*	*12%/80%/8%*	–	*10%/82%/8%*	*12%/80%/8%*	–	*10%/82%/8%*	*12%/80%/8%*
AskDifferent	**0.902**	0.873[a]	0.815[a]	**0.244**	0.227[a]	0.216[a]	**0.373**	0.351[a]	0.342[a]
(W/E/L)	–	*11%/84%/5%*	*26%/70%/4%*	–	*11%/84%/5%*	*26%/70%/4%*	–	*11%/84%/5%*	*26%/70%/4%*
Database Administrator	**0.889**	0.854[a]	0.816[a]	**0.223**	0.209[a]	0.201[a]	**357**	0.338[a]	0.323[a]
(W/E/L)	–	*14%/78%/8%*	*20%/73%/7%*	–	*14%/78%/8%*	*20%/73%/7%*	–	*14%/78%/8%*	*20%/73%/7%*
Freecode	**0.834**	0.763[a]	0.692[a]	**0.209**	0.185[a]	0.172[a]	**0.321**	0.283[a]	0.262[a]
(W/E/L)	–	*14%/76%/10%*	*24%/68%/8%*	–	*14%/76%/10%*	*24%/68%/8%*	–	*14%/76%/10%*	*24%/68%/8%*
Wordpress	**0.861**	0.832[a]	0.765[a]	**0.204**	0.187[a]	0.173[a]	**0.315**	0.297[a]	0.283[a]
(W/E/L)	–	*11%/83%/6%*	*21%/74%/5%*	–	*11%/83%/6%*	*21%/74%/5%*	–	*11%/83%/6%*	*21%/74%/5%*
Askubuntu	**0.870**	0.832[a]	0.770[a]	**0.224**	0.214[a]	0.198[a]	**0.344**	0.327[a]	0.303[a]
(W/E/L)	–	*24%/71%/5%*	*27%/68%/5%*	–	*24%/71%/5%*	*27%/68%/5%*	–	*24%/71%/5%*	*27%/68%/5%*
Serverfault	**0.861**	0.823[a]	0.708[a]	**0.234**	0.218[a]	0.196[a]	**0.357**	0.337[a]	0.304[a]
(W/E/L)	–	*21%/72%/7%*	*28%/65%/7%*	–	*21%/72%/7%*	*28%/65%/7%*	–	*21%/72%/7%*	*28%/65%/7%*
Unix	**0.877**	0.816[a]	0.722[a]	**0.229**	0.212[a]	0.182[a]	**0.344**	0.326[a]	0.282[a]
(W/E/L)	–	*18%/74%/8%*	*30%/63%/7%*	–	*18%/74%/8%*	*30%/63%/7%*	–	*18%/74%/8%*	*30%/63%/7%*
StackOverflow@large	**0.839**	0.795[a]	0.762[a]	**0.271**	0.232[a]	0.213[a]	**0.364**	0.331[a]	0.309[a]
(W/E/L)	–	*18%/77%/5%*	*27%/68%/5%*	–	*18%/77%/5%*	*27%/68%/5%*	–	*18%/77%/5%*	*27%/68%/5%*
Weighted Average	**0.841**	0.793	0.762	**0.268**	0.231	0.212	**0.363**	0.331	0.309

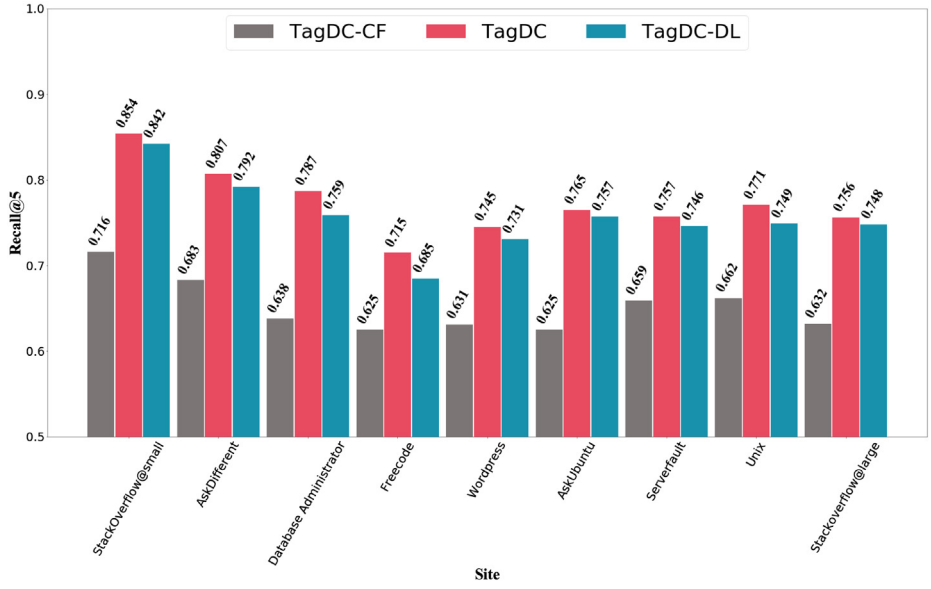
[a]Denotes that the $p - value < 0.001$ when testing the difference of TagDC and baselines.

Table 4 provides a detailed analysis of the experimental results obtained. We give an example with two software objects to understand the complementary function of TagDC-CF to TagDC-DL. It is observed that TagDC with both deep learning techniques and collaborative filtering techniques achieves more accurate results than TagDC-DL for "iterator" and "opp" in these two software objects. The reason might be that TagDC-DL depends on enough training samples, but there are a few software objects with these two tags. TagDC-CF can locate the most similar objects to the current object and use their tags for recommendation. This suggests that TagDC-CF can act as a complementary part to complete the effectiveness of TagDC-DL.

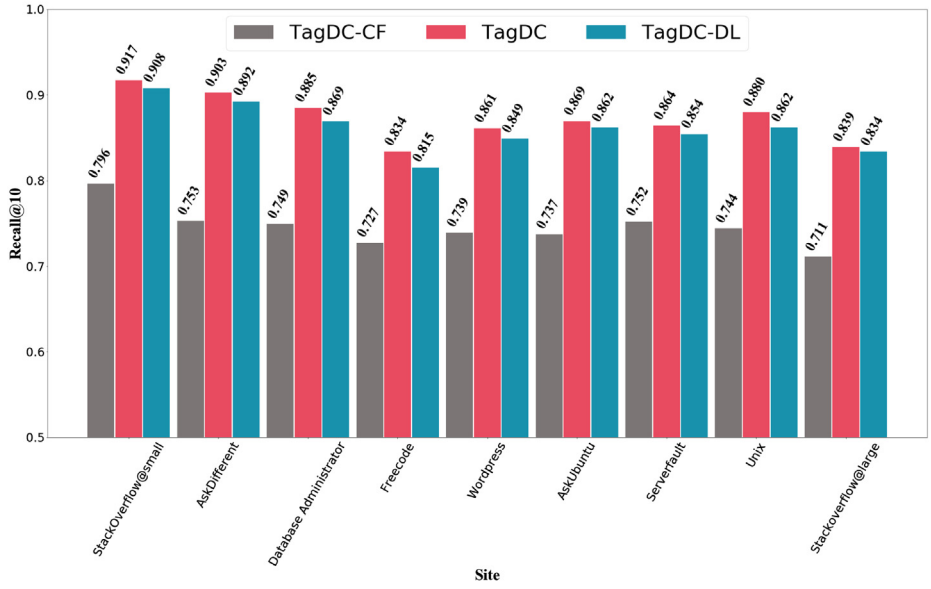
TagDC combines the advantages of deep learning and collaborative filtering techniques, while TagDC-DL affects the effectiveness most.

4.3. RQ3. How effective are different variations of TagDC-DL?

Motivation: TagDC-DL contains several components (e.g., Bi-LSTM, CNN, capsule networks). This research question aims to



(a) Recall@5



(b) Recall@10

Fig. 6. The recall comparisons of TagDC, TagDC-DL and TagDC-CF.

evaluate whether each component in the TagDC-DL module is beneficial for experimental results.

In this research question, we perform the following four variations of our TagDC-DL (see Table 5) and evaluate their contribution to the whole module on all datasets.

- TagDC-CNN: without the word representation learning process and capsule networks. TagDC-CNN is the same as TagCNN (Zhou et al., 2019).
- TagDC-WC: without the capsule network.
- TagDC-CC: without the word representation learning process.
- TagDC-DL: the whole module described in Fig. 4.

For the first two models without the capsule network, we adopt a fully-connected layer with the *sigmoid* loss function to calculate the independent confidence probability for each candidate tag. We also use binary cross-entropy to train these two models. In particular, TagDC-CNN is the same as a baseline TagCNN (Zhou et al., 2019). To visually show the differences between these variations of TagDC-DL, we use the six evaluation metrics to evaluate the effectiveness. The experimental results are shown in Table 6.

It is observed that the word representation learning process and the capsule networks are useful for improving the effectiveness on these datasets. With the word representation learning process, TagDC-DL achieves better effectiveness over TagDC-CC on all datasets, TagDC-WC outperforms TagDC-CNN also on all datasets. Similarly, the effectiveness gaps between TagDC-CNN and TagDC-CC, TagDC-WCC and TagDC-DL also show that the capsule network improves the effectiveness. Especially, TagDC-DL achieves the best effectiveness on all datasets. The experimental results of these four variations again prove that both the

Table 4
Examples of tag recommendation using TagDC-DL, TagDC-CF and TagDC on StackOverflow.

NO. 330 software object	
Title	Should I use nested classes in this case?
Body	I am working on a collection of classes used for video playback and recording. ····· Nested classes are a new concept to me. Just want to see what programmers think about the issue.
Tags	c++, oop, class, nested-class
the results of tag Recommended:	
TagDC-DL	c++, class, nested-class, c, dictionary,
TagDC-CF	c++, class, database, c, ***opp***
TagDC	c++, class, nested-class, ***opp***, c
NO. 250,874 software object	
Title	Iterator pattern in VB.NET (C# would use yield!)
Body	How do implement the iterator pattern in VB.NET, which does not have the yield keyword?
Tags	vb.net design-patterns ***iterator*** yield
the results of Tag Recommended:	
TagDC-DL	c#, .net, vb.net, lambda, design-patterns
TagDC-CF	c#, .net, c++, vb.net, ***iterator***
TagDC	c#, .net, vb.net, design-patterns, ***iterator***

Table 5
The construction of different TagDC-DL variations.

Model	Word representation learning	CNN	Capsule networks
TagDC-CNN		√	
TagDC-WC	√	√	
TagDC-CC		√	√
TagDC-DL	√	√	√

word representation learning process and the capsule networks contribute to tag recommendation tasks.

TagDC-DL can benefit from the word representation learning process and capsule networks for tag recommendation tasks.

4.4. RQ4. How efficient is TagDC?

Motivation: Tag recommendation methods are expected to recommend tags for a new posting in negligible time. As shown in Fig. 3, TagDC needs to be trained before used for tag recommendation. Reporting the time usage of model training and prediction helps us measure the efficiency of TagDC better.

To investigate the training time needed to construct our recommendation model and the prediction time to recommend tags, we record the start time and the end time of training and prediction phases. For building our TagDC, we first need to train word vectors through Word2Vec as the input of TagDC-DL and document vectors through Doc2vec as the input of TagDC-CF. Therefore, the cost of training Word2vec and Doc2vec is also covered in the training time. The experimental results are given in Table 7.

It is observed that TagDC's training time never exceeds 924 s on five small-scale datasets, 3965 s on three medium-scale datasets, and 214,819 s on one large-scale dataset. Although the training time for StackOverflow@large is relatively significant, our model can be done offline. In the prediction phase, TagDC only takes negligible time to recommend tags for each software object.

TagDC's time usage for the training and prediction phases is acceptable for various-scale datasets.

5. Discussion

In this section, we first discuss the impact of an important parameter of TagDC-CF and evaluate the effectiveness of our method on a new dataset. Then we provide the implications of the results taken from our experiments, followed by the analysis of threats to validity.

5.1. What is the impact of changing the number of TOP-N selected similar software objects on the effectiveness of TagDC-CF?

The number of similar software objects is an important parameter that affects the effectiveness of TagDC. To better reflect the changing trend of experimental results as the number of similar software objects increases, we select five small-scale datasets, including StackOverflow@small, AskDifferent, Database Administrator, Freecode, and Wordpress to conduct this group of experiments. For each dataset, we perform TagDC-CF on it and gradually increase the N value from 10 to 100 step by 10. Fig. 7(a)–7(f) respectively depict the trend of experimental results on these five datasets in terms of *Recall*@5, *Precision*@5, $F1-score$@5, *Recall*@10, *Precision*@10 and $F1-score$@10 when varying the number of TOP-N similar software objects.

From the experimental results, we can see that the parameter N affects evaluation results. For all five datasets, As the value of N increases, the experimental results in terms of these six evaluation metrics first steadily rise, then the growth rate becomes

Table 6
Recall@k, Precision@k, and F1-score@k of different TagDC variations.

Site	Model	Recall@5	Precision@5	F1-score@5	Recall@10	Precision@10	F1-score@10
StackOverflow@small	TagDC-CNN	0.812	0.343	0.463	0.890	0.191	0.305
	TagDC-WC	0.823	0.347	0.466	0.894	0.193	0.308
	TagDC-CC	0.831	0.349	0.469	0.899	0.193	0.310
	TagDC-DL	0.842	0.354	0.476	0.908	0.195	0.313
AskDifferent	TagDC-CNN	0.773	0.404	0.504	0.873	0.227	0.351
	TagDC-WC	0.780	0.409	0.510	0.878	0.230	0.356
	TagDC-CC	0.784	0.412	0.514	0.881	0.232	0.358
	TagDC-DL	0.792	0.419	0.521	0.892	0.238	0.364
Database Administrator	TagDC-CNN	0.736	0.354	0.462	0.854	0.209	0.338
	TagDC-WC	0.742	0.359	0.464	0.857	0.211	0.341
	TagDC-CC	0.748	0.364	0.468	0.860	0.215	0.344
	TagDC-DL	0.759	0.369	0.476	0.869	0.218	0.348
Freecode	TagDC-CNN	0.644	0.306	0.389	0.763	0.185	0.283
	TagDC-WC	0.657	0.311	0.394	0.772	0.188	0.287
	TagDC-CC	0.668	0.317	0.404	0.802	0.193	0.295
	TagDC-DL	0.685	0.327	0.415	0.815	0.196	0.299
Wordpress	TagDC-CNN	0.707	0.312	0.414	0.832	0.187	0.297
	TagDC-WC	0.713	0.315	0.417	0.836	0.191	0.300
	TagDC-CC	0.718	0.322	0.420	0.841	0.194	0.304
	TagDC-DL	0.731	0.332	0.431	0.849	0.199	0.312
AskUbuntu	TagDC-CNN	0.724	0.360	0.465	0.832	0.214	0.327
	TagDC-WC	0.738	0.364	0.469	0.846	0.217	0.330
	TagDC-CC	0.744	0.367	0.472	0.851	0.219	0.332
	TagDC-DL	0.757	0.377	0.478	0.862	0.221	0.338
Severefault	TagDC-CNN	0.699	0.364	0.457	0.823	0.218	0.337
	TagDC-WC	0.722	0.372	0.471	0.835	0.223	0.343
	TagDC-CC	0.735	0.377	0.477	0.843	0.226	0.347
	TagDC-DL	0.746	0.383	0.485	0.854	0.229	0.350
Unix	TagDC-CNN	0.685	0.337	0.430	0.816	0.212	0.326
	TagDC-WC	0.711	0.346	0.445	0.838	0.218	0.330
	TagDC-CC	0.732	0.357	0.456	0.850	0.222	0.333
	TagDC-DL	0.749	0.362	0.464	0.862	0.224	0.335
StackOverflow@large	TagDC-CNN	0.701	0.392	0.501	0.795	0.232	0.331
	TagDC-WC	0.722	0.404	0.515	0.811	0.248	0.346
	TagDC-CC	0.735	0.409	0.524	0.823	0.259	0.354
	TagDC-DL	0.748	0.415	0.532	0.834	0.268	0.360

Table 7
The training and prediction time of TagDC on nine datasets.

Sites	Training time (s)	Prediction time (ms)
StackOverflow@small	624	7.525
AskDifferent	924	11.480
Database Administrator	873	11.442
Freecode	565	6.644
Wordpress	846	9.221
Askubuntu	3,965	13.732
Serverfault	3,666	14.741
Unix	1,689	10.849
StackOverflow@large	214,819	783.623

slow. When N reaches a certain value, the results begin to decrease. That is because noise is inevitably introduced considering software objects with low similarity to the current recommended object. Therefore, We conclude that an appropriate number of similar software objects is to be preferred.

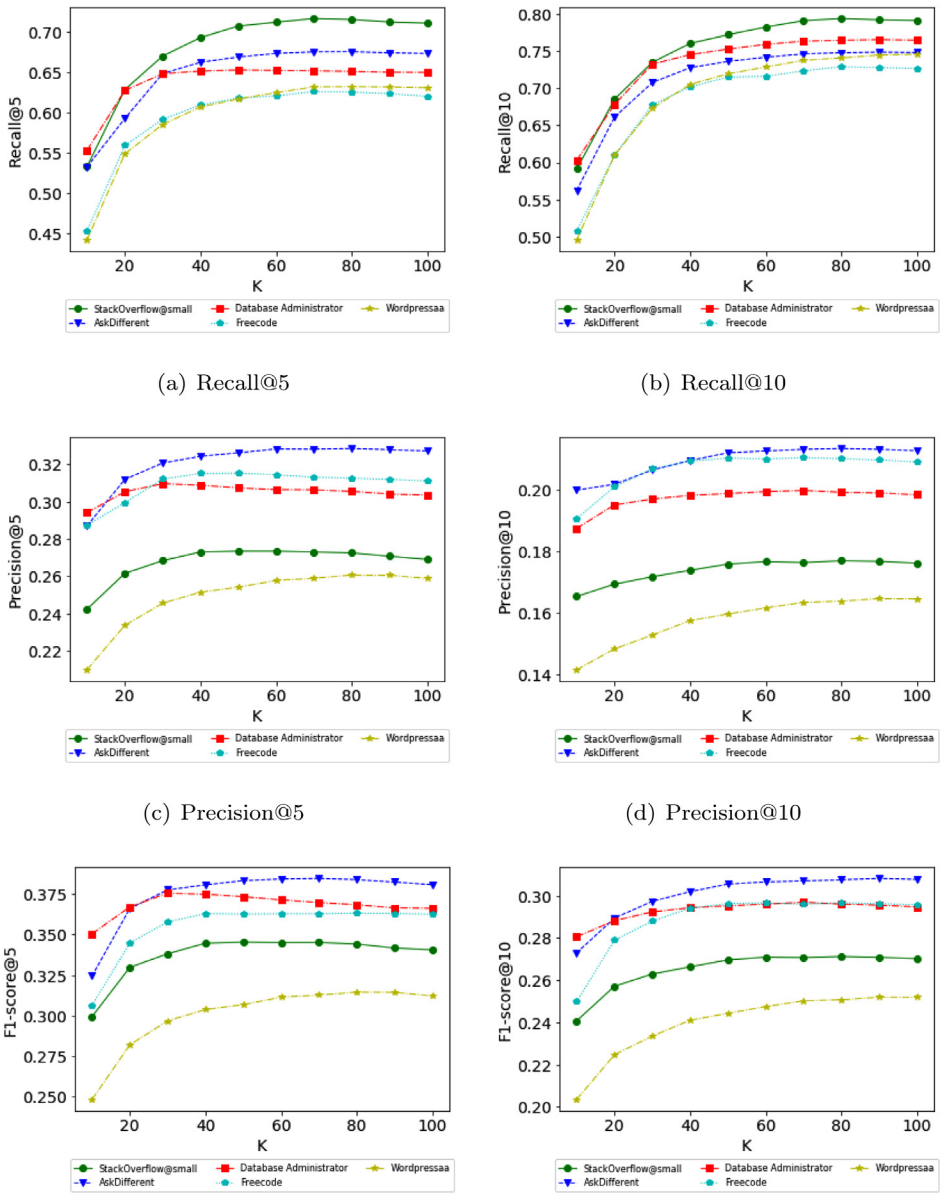
5.2. How our method performs on new data?

To evaluate whether our TagDC can perform well on relatively new software objects, we construct a new dataset called StackOverflow@new by collecting the postings from January 1st, 2020 to March 1st, 2020. The older data is relatively unlikely to be modified again. Therefore, we set the deadline for March 1st, 2020 to ensure the stability of the data. The details of StackOverflow@new are in Table 8. We implement our method and two baselines on this dataset with the same experimental settings. The experimental results are shown in Table 9.

It is observed in Table 9 that TagDC achieves higher *Recall@k*, *Precision@k*, and $F1-score@k$ values than two baselines on StackOverflow@new. This confirms the effectiveness of our method on recently posted software objects.

5.3. Implications of our work

Implications for researchers: The key implication for researchers is to take inspiration from our automatic tag recommendation method. To the best of our knowledge, postings in the software information sites is a hot research topic in the field



(a) Recall@5 (b) Recall@10

(c) Precision@5 (d) Precision@10

(e) F1-score@5 (f) F1-score@10

Fig. 7. The effect of the number of TOP-N selected software objects for TagDC-CF.

Table 8
Statistics of StackOverflow@new.

Site name	Software objects	Tags	Final Software objects	Final tags
StackOverflow@new	267,804	16,708	264,624	1556

Table 9
TagDC vs. TagCNN & FastTagRec on StackOverflow@new.

Model	Recall@5	Precision@5	F1-score@5	Recall@10	Precision@10	F1-score@10
TagDC	0.798	0.380	0.494	0.882	0.218	0.336
TagCNN	0.732	0.349	0.454	0.828	0.204	0.318
FastTagRec	0.686	0.328	0.426	0.781	0.193	0.298

of software engineering. Thus, the feature extraction of postings is significantly important. Our method is a useful attempt to improve the capacity of feature extraction for postings by combining both deep learning techniques and collaborative filtering techniques. Deep learning techniques can effectively extract rich semantics features from postings. For example, the Bi-LSTM model can extract long-distance semantics, while the CNN model

and capsule networks can extract consecutive semantics. Considering the consecutive semantics and long-distance semantics of a posting simultaneously helps to mine the key information more accurately. The collaborative filtering techniques take advantage of the similarities among postings for more accurate semantics learning. The key to our method is to improve the capacity of feature extraction of postings. Therefore, we believe that some researches about postings (e.g., summarizing the topics of postings, answers tagging, etc.) can benefit from our method.

Implications for practitioners: The key implication for practitioners is to apply TagDC to help them tag new postings. For a mature software information site, the topics of the new postings are possibly associated with existing popular tags. TagDC collects these tags as the candidate tag set, and recommend several tags from the candidate tag set for new postings. The tags recommended by TagDC can provide a reference for developers when they select tags for new postings. This not only helps users more easily select the appropriate tags, but also reduce the creation of new tags. Thus, we think that TagDC can facilitate the tagging process of developers.

5.4. Threats to validity

In this subsection, we analyze several potential threats to the validity of our experimental results from different aspects.

Internal Threats: The internal threats are related to the reliability of our selected datasets. Because developers are free to create tags in their own words, errors are inevitably introduced. To alleviate the problem, we apply the same strategies as past tag recommendation tasks (Wang et al., 2015; Zhou et al., 2017; Liu et al., 2018; Zhou et al., 2019). For example, because the older data is relatively unlikely to be modified again, we conduct our experiments on the widely used software information sites and choose relatively older software objects to ensure the stability of experimental data. We also filter out the tags rarely appear and software objects whose tags are all low-frequency. Furthermore, we have designed a case study to determine the ground truth of our experimental data. To be consistent with past tag recommendation tasks, we use the same datasets including Freecode. However, Freecode is an old dataset that has not been updated since June 2014. Using this old dataset may be a potential threat to the validity of our experimental results.

External Threats: The external threats refer to the generality of the proposed method. We have deployed TagDC on eight software information sites with more than 11 million software objects. In future work, more datasets will be selected for evaluation. Our current work only considers popular tags in the candidate tag set. For unpopular tags, it is not easy to directly use our approach. We need to consider their corresponding software objects as training data to retrain our model when there are enough software objects.

6. Related work

Many automatic tag recommendation approaches have also been proposed in the field of software engineering recently. TagRec was first proposed by Al-Kofahi et al. (2010) to automatically infer several tags for work items in IBM Jazz. It is based on the fuzzy set theory and considered the dynamic evolution of a system. Later Wang et al. (2015) proposed TagCombine to recommend tags for software objects on software information sites. TagCombine is composed of three modules: a multi-label ranking module, a similarity module, and a tag-term module. However, the multi-label ranking module in TagCombine converts the multi-label classification task into many binary classification. It has to train thousands of binary classifier models for large-scale software information sites such as StackOverflow@large. Therefore, TagCombine is limited in relatively small software information sites. Then Wang et al. proposed (Wang et al., 2014) EnTagRec, including two modules (Bayesian inference module and Frequentist inference module), which outperformed the effectiveness of TagCombine. However, it is not a scalable method because it depends on all information in a site to train a model. Several years later, Wang et al. (2018) improved the original EnTagRec and put forward EnTagRec++ by taking consideration of users' information to improve the accuracy of tag recommendation task. A more advanced method TagMulRec was proposed by Zhou et al. (2017), which built indices for the descriptions extracted from software objects and ranks the scores of all candidate tag for tag recommendation. However, TagMulRec only utilizes a small part of information as only several most similar software objects to the given software object in a software information site are taken into account. Then a tag recommendation method based on topic modeling approaches was proposed by Hong et al. (2017). It recommends tags for software objects by computing tag scores based on both the document similarities and the occurrence of historical tags. Lately, Liu et al. put forward FastTagRec based on a single hidden layer neural network (Liu et al., 2018), which was not only scalable and accurate, but also faster than the existing approaches. It adopts the semantic relationship between historical software objects and their responding tags to perform the tag recommendation task. Sonam et al. (2019) proposed the TagStack system, a machine learning and feedback-based method to recommend tags on StackOverflow. Then, Zhou et al. (2019) discussed the advantage of deep learning methods over traditional machine learning methods in the field of tag recommendation. They found that TagCNN achieved the state-of-the-art results.

Similarly, tags studies have been a hot research topic in the software engineering field in recent years. Eynard et al. (2013) found the intrinsic advantages of Tag-based systems and the inherent lexical ambiguities of tags. They analyzed the label-based system and proposed a theoretical basis for solving tag synonyms. Treude and Storey (2009) carry out an empirical research for a large project to study how tagging had been adopted and adapted in the last two years. They concluded that the tagging mechanism provided significant help in bridging the gap between technical and social aspects of organizing work items. Later Thung et al. (2012) implemented a user study with several related participants. The research results show that the collaborative tagging mechanism was promising to detect similar software applications as a useful information source. Wang et al. (2012) performed a group of experiments on the Freecode site to research the semantic relationship among tags, and defined the relationship as a taxonomy. To utilize the relationship among tags to alleviate the rapid growth of tags, Beyer and Pinzger (2015) investigated the strategy to build synonym tag pairs for StackOverflow and developed a tag synonym recommendation method TSST by implementing this strategy. Beyer and Pinzger (2016) continued their previous research and proposed a new approach to classify tag synonyms into various meaningful topics. Saleh and El-Tazi (2017) proposed a method based on the topic model, which generated tag groups by capturing topics of documents and classifying the top tags associated with documents. Chen et al. (2019) created a hierarchical organization based on the tags in StackOverflow to help categorize the vast and growing contents posted on this software information site.

7. Conclusion and future work

In this paper, we propose a novel composite model TagDC with two modules (i.e., TagDC-DL and TagDC-CF) for tag recommendation task. Our proposed method leverages the information

from the description of each software object. It combines the advantages of deep learning techniques and collaborative filtering techniques. In detail, TagDC-DL constructs a multi-label classifier by learning from the historical software objects and their corresponding tags. TagDC-CF is a complementary part of TagDC-DL. It can enhance our model's effectiveness by locating the most similar software objects to the current object. Extensive experiments are conducted on nine datasets to evaluate TagDC from different respects. The experimental results show that TagDC substantially improves state-of-the-art methods for the tag recommendation task.

In the future, we plan to take additional features (e.g., code snippets, screenshots, etc.) into consideration for a comprehensive representation of each object. In the current model, we use the document vector of each description to measure the cosine similarities among software objects. There may exist more appropriate input for TagDC-CF, which will be explored in our future work. When combining two modules, we use a linear combination strategy. The other combination strategies will be attempted in the future. The method will be popularized to more software information sites. Furthermore, our current work cannot recommend unpopular tags that are not in the candidate tag set directly. In the future, we plan to explore the solution for recommending unpopular tags.

8. Reproducibility

Our experimental datasets, codes are available via: https://github.com/kingtiger96/TagDC.

CRediT authorship contribution statement

Can Li: Writing - original draft, Methodology, Data curation. **Ling Xu:** Methodology, Software, Visualization. **Meng Yan:** Supervision, Formal analysis, Conceptualization. **Yan Lei:** Writing - review & editing.

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.

Acknowledgments

The work described in this paper was partially supported by the National Natural Science Foundation of China (Grant no. 61772093), the National Key Research and Development Project, China (Grant no. 2018YFB2101203), the Fundamental Research Funds for the Central Universities, China (Grant no. 2019CDYGYB014).

References

Al-Kofahi, J.M., Tamrawi, A., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N., 2010. Fuzzy set approach for automatic tagging in evolving software. In: 2010 IEEE International Conference on Software Maintenance. IEEE, pp. 1–10.

Barua, A., Thomas, S.W., Hassan, A.E., 2014. What are developers talking about? an analysis of topics and trends in stack overflow. Empir. Softw. Eng. 19 (3), 619–654.

Beyer, S., Pinzger, M., 2015. Synonym suggestion for tags on stack overflow. In: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension. IEEE Press, pp. 94–103.

Beyer, S., Pinzger, M., 2016. Grouping android tag synonyms on stack overflow. In: Proceedings of the 13th International Conference on Mining Software Repositories. ACM, pp. 430–440.

Bird, S., Klein, E., Loper, E., 2009. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media, Inc.

Chen, H., Coogle, J., Damevski, K., 2019. Modeling stack overflow tags and topics as a hierarchy of concepts. J. Syst. Softw. 156, 283–299.

Costa, C., Figueiredo, J., Murta, L., Sarma, A., 2016. TIPMerge: recommending experts for integrating changes across branches. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, pp. 523–534.

Eynard, D., Mazzola, L., Dattolo, A., 2013. Exploiting tag similarities to discover synonyms and homonyms in folksonomies. Softw. - Pract. Exp. 43 (12), 1437–1457.

Hong, B., Kim, Y., Lee, S.H., 2017. An efficient tag recommendation method using topic modeling approaches. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems. ACM, pp. 56–61.

Joorabchi, A., English, M., Mahdi, A.E., 2015. Automatic mapping of user tags to Wikipedia concepts: The case of a Q&A website–StackOverflow. J. Inf. Sci. 41 (5), 570–583.

Kim, J., Jang, S., Choi, S., Park, E., 2018. Text classification using capsules. arXiv preprint arXiv:1808.03976.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Lai, S., Xu, L., Liu, K., Zhao, J., 2015. Recurrent convolutional neural networks for text classification. In: Twenty-Ninth AAAI Conference on Artificial Intelligence.

Lau, J.H., Baldwin, T., 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. arXiv preprint arXiv:1607.05368.

Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196.

Liu, J., Zhou, P., Yang, Z., Liu, X., Grundy, J., 2018. FastTagRec: fast tag recommendation for software information sites. Autom. Softw. Eng. 25 (4), 675–701.

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

Peng, H., Li, J., Gong, Q., Wang, S., He, L., Li, B., Wang, L., Yu, P.S., 2019. Hierarchical taxonomy-aware and attentional graph capsule RCNNs for large-scale multi-label text classification. arXiv preprint arXiv:1906.04898.

Rodríguez, P., Bautista, M.A., Gonzalez, J., Escalera, S., 2018. Beyond one-hot encoding: Lower dimensional target embedding. Image Vis. Comput. 75, 21–31.

Saleh, I., El-Tazi, N., 2017. Automatic organization of semantically related tags using topic modelling. In: European Conference on Advances in Databases and Information Systems. Springer, pp. 235–245.

Sonam, S., Verma, A., Lal, S., Sardana, N., 2019. TagStack: Automated system for predicting tags in stackoverflow. In: 2019 International Conference on Signal Processing and Communication, ICSC, pp. 223–228.

Thung, F., Lo, D., Jiang, L., 2012. Detecting similar applications with collaborative tagging. In: 2012 28th IEEE International Conference on Software Maintenance. ICSM, IEEE, pp. 600–603.

Treude, C., Storey, M.-A., 2009. How tagging helps bridge the gap between social and technical aspects in software development. In: Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, pp. 12–22.

Wang, S., Lo, D., Jiang, L., 2012. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In: 2012 28th IEEE International Conference on Software Maintenance. ICSM, IEEE, pp. 604–607.

Wang, S., Lo, D., Vasilescu, B., Serebrenik, A., 2014. EnTagRec: An enhanced tag recommendation system for software information sites. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 291–300.

Wang, S., Lo, D., Vasilescu, B., Serebrenik, A., 2018. EnTagRec++: An enhanced tag recommendation system for software information sites. Empir. Softw. Eng. 23 (2), 800–832.

Wang, X.-Y., Xia, X., Lo, D., 2015. Tagcombine: Recommending tags to contents in software information sites. J. Comput. Sci. Tech. 30 (5), 1017–1035.

Wilcoxon, F., Individual comparisons by ranking methods. In: Kotz, S., Johnson, N.L. (Ed.), Breakthroughs in Statistics: Methodology and Distribution, pp. 196–202.

Williams, R.J., Peng, J., 1990. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. Neural Comput. 2 (4), 490–501.

Xiao, L., Zhang, H., Chen, W., Wang, Y., Jin, Y., 2018. Mcapsnet: Capsule network for text with multi-task learning. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 4565–4574.

Zangerle, E., Gassler, W., Specht, G., 2011. Using tag recommendations to homogenize folksonomies in microblogging environments. In: International Conference on Social Informatics. Springer, pp. 113–126.

Zhang, K., Zuo, W., Gu, S., Zhang, L., 2017. Learning deep CNN denoiser prior for image restoration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3929–3938.

Zhao, W., Ye, J., Yang, M., Lei, Z., Zhang, S., Zhao, Z., 2018. Investigating capsule networks with dynamic routing for text classification. arXiv preprint arXiv:1804.00538.

Zhou, P., Liu, J., Liu, X., Yang, Z., Grundy, J., 2019. Is deep learning better than traditional approaches in tag recommendation for software information sites? Inf. Softw. Technol. 109, 1–13.
Zhou, P., Liu, J., Yang, Z., Zhou, G., 2017. Scalable tag recommendation for software information sites. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 272–282.

Can Li is a master student in the School of Big Data & Software Engineering, Chongqing University, China. His research interests include intelligent software engineering, feature engineering, data mining.

Ling Xu is an Associate Professor at the School of Big Data & Software Engineering, Chongqing University, China. She received her B.S. degree in Hefei University of Technology in 1998, and her M.S. degree in software engineering in 2004. She received her Ph.D. degree in Computer Application from Chongqing University, P.R. China in 2009. Her research interests include mining software repositories, bug reduction, and localization.

Meng Yan is now an Assistant Professor at the School of Big Data & Software Engineering, Chongqing University, China. Prior to joining Chongqing University, he was a Postdoc at Zhejiang University advised by Prof. Shanping Li and Dr. Xin Xia. He got his Ph.D. degree in June 2017 under the supervision of Prof. Xiaohong Zhang from Chongqing University, China. His current research focuses on how to improve developers' productivity, how to improve software quality, and how to reduce the effort during software development by analyzing rich software repository data.

Yan Lei received the B.A., M.A., and Ph.D. degrees in computer science and technology, all from the National University of Defense Technology, China. He is an associate professor at School of Big Data & Software Engineering, Chongqing University, China. His research interests include fault localization program repair, program slicing, etc.