FastTagRec: Fast Tag Recommendation for Software Information Sites

Pingyi Zhou · Jin Liu · Zijiang Yang · Xiao Liu · John Grundy

Accepted 27th June 2018

Abstract Software information sites such as `StackOverflow` and `Freeecode` enable information sharing and communication for developers around the world. To facilitate correct classification and efficient search, developers need to provide tags for their postings. However, tagging is inherently an uncoordinated process that depends not only on developers' understanding of their own postings but also on other factors, including developers' English skills and knowledge about existing postings. As a result, developers keep creating new tags even though existing tags are sufficient. The net effect is an ever increasing number of tags with severe redundancy along with more postings over time. Any algorithms based on tags become less efficient and accurate. In this paper we propose FastTagRec, an automated scalable tag recommendation method using neural network-based classification. By learning existing postings and their tags from existing information, FastTagRec is able to very accurately infer tags for new postings. We have implemented a prototype tool and carried out experiments on ten software information sites. Our results show that Fast-

Pingyi Zhou
State Key Lab. of Software Engineering, Computer School, Wuhan University, China.
E-mail: zhou_pinyi@whu.edu.cn

Jin Liu
State Key Lab. of Software Engineering, Computer School, Wuhan University, China.
E-mail: jinliu@whu.edu.cn

Zijiang Yang
Department of Computer Science, Western Michigan University, Kalamazoo, Michigan, USA.
E-mail: zijiang.yang@wmich.edu

Xiao Liu
School of Information Technology, Deakin University, Melbourne, Australia.
E-mail: xiao.liu@deakin.edu.au

John Grundy
School of Information Technology, Deakin University, Melbourne, Australia.
E-mail: j.grundy@deakin.edu.au

TagRec is not only more accurate but also three orders of magnitude faster than the comparable state-of-the-art tool TagMulRec. In addition to empirical evaluation, we have also conducted an user study which successfully confirms the usefulness of of our approach.

Keywords Software Information Site · Software Object · Tag Recommendation ·

1 Introduction

Community-based Question Answering (cQA) services and Community-based Open Source (cOS) services provide a valuable online resource for developers around the world. These online platforms – called **software information sites** (Xia et al 2013; Wang et al 2014; Zhou et al 2017) – help developers on all kinds of issues across the whole life cycle of software development. Well-known software information sites include `StackOverflow` (`http://www.stackoverflow.com`) and `Freecode` (`http://www.freecode.com`). Due to their importance, software information sites have attracted great attention from both academia and industry (Treude and Robillard 2016; Robillard and Medvidović 2016; Fowkes and Sutton 2016; Michaud et al 2016; Xia et al 2014; Hou and Mo 2013). For example, there have been efforts to search duplicate postings (Hindle et al 2016; Thung et al 2014) and unsolved questions (Zhao et al 2015; Yang et al 2013).

The developer-generated content of these software information sites – such as a question with answers in a developer Q&A site and a project in a developer open source site – are termed **software objects** (Xia et al 2013; Wang et al 2014; Zhou et al 2017). A software object in a developer Q&A site, such as `StackOverflow`, includes title, body, tags, comments, etc. Title and body give concise and detailed descriptive information about a question. Figure 1 shows a question in `StackOverflow` with title `how to add pattern validation in angular` at the top, question description at the middle and five tags `c#, html, angularjs, asp.net-mvc, angularjs-directive` at the bottom. The component between title and tags is the body.

Similarly, an open source project in a developer open source site, such as `Freecode`, includes a project name, project description, various tags, etc. Figure 2 shows an open source project shared in `Freecode` with project name `QuartzDesk` at the top, project description in the middle, and four example tags `Quartz Scheduler, management, Monitoring, Web Application` at the bottom.

No matter if it is a developer Q&A site or a developer open source site, a software information site usually requires developers to classify software objects with multiple tags at the time of posting. These tags are regarded as an efficient and lightweight computing mechanism in promoting developers' communication, assiting in information finding, and helping reduce the gap between social and technical aspects (Treude and Storey 2009). Tags provide critical metadata to search, describe, identify, bookmark, classify, and organize

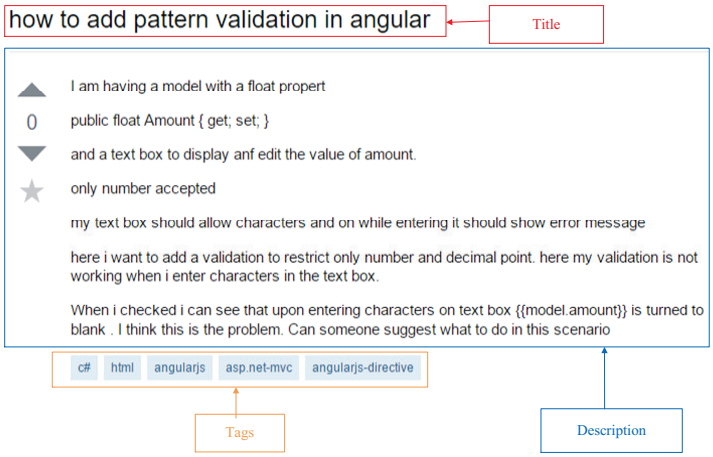


Fig. 1: A question posted on `StackOverflow`

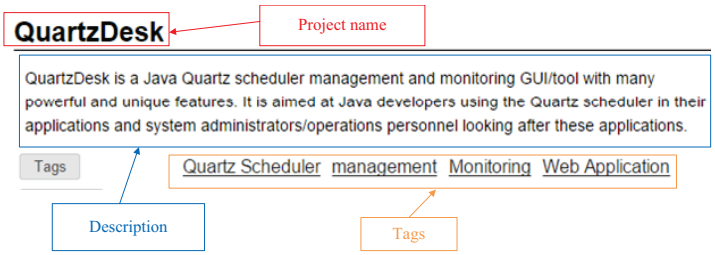


Fig. 2: An open source project shared on `Freecode`

software objects in these software information sites (Al-Kofahi et al (2010); Zhou et al (2017)). Therefore, the overall quality and usefulness of software information sites depends on high quality tags that concisely describe the most important features of the hosted software objects.

Unfortunately, tagging software objects by developers is inherently a distributed and uncoordinated process. Most software information sites allow developers to tag their software objects with their own words. Because of the freedom, tags can be idiosyncratic due to developer's understanding of their software objects, English skills and preferences. For example, in `StackOverflow` the tags `oop`, `oo`, `ood`, `object-orientation` and several other words are all used to describe object-oriented programming. This phenomenon is called tag synonyms (Xia et al 2013; Wang et al 2014). In addition, each software object is usually accompanied by multiple tags. For example, developers are asked to label at least 3 and no more than 5 tags per posting in `StackOverflow`. `Freecode` allows developers to attach more than ten tags per sharing. As a result the number of different tags grows rapidly along with continuous addition of software objects. So far, `StackOverflow` has more than 20 million ques-

tions and 46 thousand tags. If software information sites continue the current practice of free tag choices by developers, the number of tags will continue to grow. With such a large number of tags, tagging eventually loses its capability to help maintain a useful software information site.

In order to better tag new content, effectively reuse existing tags, and efficiently manage the growth of tags in a software information site, we propose a new automated tag recommendation tool called FastTagRec. The assumption is that for a mature and large software information site, the topics of a newly posted software objects are very likely to be covered by the vast number of existing software objects. However, this fact is largely ignored because it is easier for developers to create their own tags than to browse and select existing tags. For example, a developer is likely to create a tag `oo` for a software object related to object-oriented programming rather than to make an effort on searching for tags that have already been used for this topic. FastTagRec learns from existing software objects and their tags, and recommends appropriate tags for new software objects. The learning is based on natural language descriptions of each software object and the tags that are associated with the software object. In order to make FastTagRec scalable, we exploit a neural network approach that is based on single-hidden layer neural network and the rank constraint of word. FastTagRec significantly improves the state-of-the-art comparable tool TagMulRec (Zhou et al 2017). FastTagRec achieves the same goal with a completely different algorithm. Compared with TagMulRec, FastTagRec is not only more accurate but also three orders of magnitude faster. In order to evaluate FastTagRec in a realistic setting, five developers are first asked to compare the recommended tags with the ground truth on StackOverflow over 100 postings. Then, we ask five developer to tag 150 popular GitHub projects and rate the usefulness of using FastTagRec to tag these projects. The user study confirms that FastTagRec is a valuable tool and can provide more useful tags for developers.

This paper makes the following two key novel contributions.

- We propose a new tag recommendation approach that learns from existing text descriptions and tags of existing software objects. Our algorithm is scalable enough to handle very large software information sites.
- We have implemented FastTagRec and conducted experiments on ten software information sites. The experiments show that FastTagRec significantly outperforms the state-of-the-art tool TagMulRec in terms of both accuracy and efficiency.

The rest of this paper is organized as follows: Section 2 presents the motivation to our work. The technical details of FastTagRec are described in section 3, followed by experimental evaluation and the user study in Section 4. In Section 5, we share some of the important lessons that we learned in implementing our work, and discuss threats to the validity of our study. Section 6 reviews the related work. Finally Section 7 concludes the paper.

2 Motivation

Our work is motivated from two perspectives: services for developers and services for software information sites.

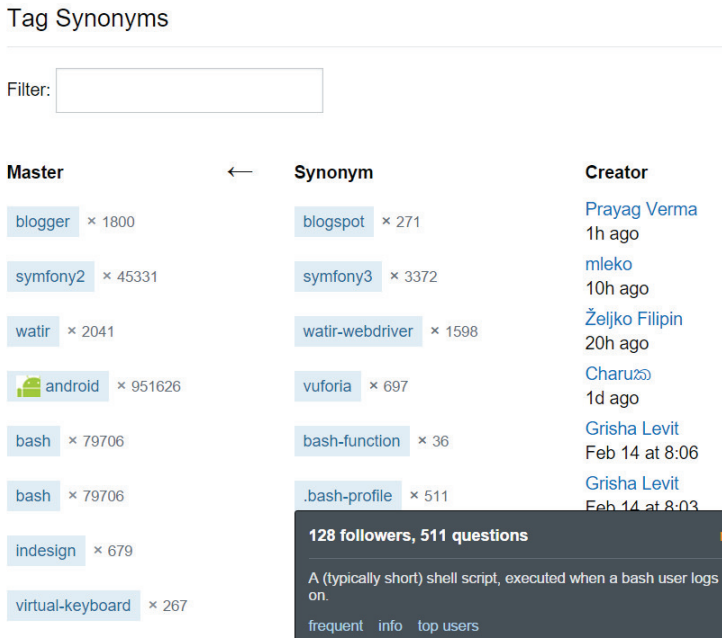


Fig. 3: Tag synonyms in StackOverflow.

2.1 Services For Developers

Since developers are free to choose tags, the words used for tags are often very arbitrary. Even for the words that represent the same meaning, there are differences such as spaces vs. no spaces, upper case vs. lower case, acronym or partial acronym vs. full spelling, hyphens vs. no hyphens, etc. Such phenomena makes it difficult for software developers to search for existing tags and thus they become more likely to use their own wording. This leads to more synonymous tags with different spelling (Beyer and Pinzger 2015). Figure 3 gives a small portion of the tag synonymous list in `StackOverflow` that contains 3429 tags. In the synonymous tag list, the master tag has a higher frequency of use than its synonymous tags, accordingly, the master tag has higher probability of being recommended to developers than its synonymous tags. So, our automated tag recommendation tool can alleviate the problem of tag synonyms. The synonymous tag list is maintained manually in `StackOverflow`, which is

very time consuming. There are some recent works on how to identify these synonymous tags in information sites(Beyer and Pinzger 2015, 2016). We can add a pre-processing step which can identify all these synonymous tags and remove them in our future work. ~~For a new developer-generated software object, FastTagRec recommends a master tag that is widely used.~~

A developer may want to utilize existing tags. However, due to the fact that the number of tags in mature software information sites is very large, it is very difficult for developers who are not familiar with existing tags to select appropriate ones to label a new software object. FastTagRec alleviates this issue by searching existing tags for developers based on natural language-based descriptions of the new software object.

2.2 Services For Software Information Sites

For software information sites, tags are used to identify, classify, and organize software objects in these platforms. If developers label software objects with similar concerns, goals, subjects or functions with different tags, the management of the software information sites become less efficient and less accurate. For an evolving large-scale software information site such as `StackOverflow`, the organization of software objects is crucial to the search speed. If we can ideally multi-classify the posted contents or shared projects, the management of software information sites will be more efficient and the response time to developers' inquires will be significantly improved.

3 Our Approach For Automated Tag Recommdation

3.1 Problem Formulation

A software information site is a set $S = \{o_1, \ldots, o_n\}$, where $o_i (1 \leq i \leq n)$ denotes a software object. For a developer Q&A site, the attributes of o_i include an identifier $o_i.id$, a body $o_i.b$, a title $o_i.tt$, and a set of tags $o_i.T$. For a developer open source site, the attribute of o_i include project name $o_i.n$, project description $o_i.b$ and a set of tags $o_i.T$. If we treat the combination of the title $o_i.tt$ and the body $o_i.b$ of a software object in a developer Q&A site as a project description $o_i.d$, we can assume that any software object o_i contain a description $o_i.d$ and a set of tags $o_i.T$. The tags in a software information site S is a set $\mathcal{TA} = \{t_1, \ldots, t_m\}$ and the tags associated with an object o_i, i.e. $o_i.T$, is a subset of $\mathcal{TA}$.

The key research question we try to answer in this paper is the following: `Given a large set of existing software objects that are tagged, how can we automatically recommend a set of appropriate tags for a new software object` o_i`?`

3.2 Approach Overview

Figure 4 depicts the overall tag recommendation architecture of FastTagRec that consists of three layers: input layer, hidden layer and output layer. This architecture is similar to the continuous bag of words model (CBOW) (Mikolov et al 2013).

In the input layer, there are N n-gram features $(f_1, \ldots, f_N)$ that are used to represent the text description $o_i.d$ of a software element. A feature in a text description can be a word in the text. For example, the two 2-gram features of the sentence"Jack loves Jane" is f_1 =(Jack loves) and f_2 =(loves Jane). In the hidden layer, these n-gram features $(f_1, \ldots, f_N)$ are converted to $(x_1, \ldots, x_N)$ and averaged to form the hidden variable X_h. Last, we use the softmax function (Bishop 2006) to compute the probability distribution $p(t_i|o_i.d)$ over the existing tags. For a set $|S|$ of tagged software objects in a software information site, this maximizes the log-likelihood over their tags:

$$\ell = \frac{\sum_{i=1}^{|s|} \sum_{j=1}^{|o_i.T|} \log p(t_j|o_i.d)}{|S|} \quad (1)$$

where t_j is a tag of software object o_i and $o_i.d$ is the text description of software object o_i. Below we describe each layer of our approach in detail.

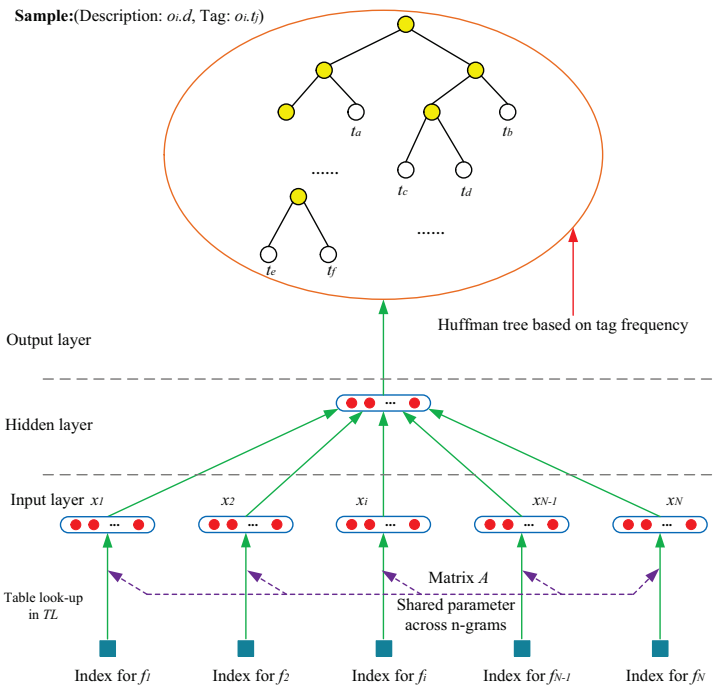


Fig. 4: The Overall Architecture of FastTagRec.

3.3 Input Layer

Given a textual, natural language-based description of a software object, a common representation for text processing and analyzing is the bag of words model (BoW) (Behley et al 2013). However, the BoW model does not consider word order. On the other hand, if we consider word order, the bag of words model is too expensive. In this paper, we use a bag of n-grams as additional features to capture partial information about the local word order.

Consider two sentences 'Jack loves Jane' and 'Jane loves Jack'. The features given by BoW model of are the same ('Jack', 'loves', 'Jane'), and thus the two sentence cannot be distinguished by the BoW model. But for a bag of bi-grams, the features of the first and the second sentences are ('Jack loves', 'loves Jane') and ('Jane loves', 'loves Jack'), respectively. Thus the two sentences can be easily distinguished. This achieves comparable results to methods that explicitly use the local word order (Wang and Manning 2012). To maintain a fast and memory efficient mapping of the n-grams, we use the hashing trick that is similar to the hash function proposed by Mikolov et. al. (Mikolov et al 2011).

FastTagRec first constructs a dictionary D that contains all words appearing in the description of software objects in a given software information site. It then constructs a look-up table TL to store the m dimension feature vectors of n-grams. The size of the look-up table is $|D|^n$ in theory, where $|D|$ is the size of D. But, many n-grams which are the combination of n words don't appear in the software information site. The hashing trick is introduced in our approach. We first scan the software information site and all n-grams appearing in the software information site is assigned a m dimension feature vector in TL. For each n-gram in TL, we randomly initialize m dimension feature vector and can locate a n-gram by index $index(n\text{-gram}) = hashcode(n\text{-gram})$. Using a hash function, we can quickly find the feature vector ft_i of a n-gram feature f_i and use less memory. To avoid the disadvantages of TagMulRec's approach, which have been discussed in the related work, a shared weight matrix A is constructed in FastTagRec. Finally, the feature vector x_i of the n-gram feature f_i can be obtained by Equation 2:

$$x_i = A \times TL(hashcode(f_i)) \in \mathbb{R}^m \tag{2}$$

In summary, given the text description of a software object, FastTagRec constructs N n-gram feature $(f_1, \ldots, f_N)$. For each n-gram feature f_i, the feature vector ft_i can be obtained by using a look-up table TL. With the help of the shared parameter of the weight matrix A, FastTagRec finally gets the n-gram feature vectors $(x_1, \ldots, x_N)$ that are used to represent a text description in the input layer.

3.4 Hidden Layer

In the hidden layer, FastTagRec computes the average of n-gram feature vectors $(x_1, \ldots, x_N)$ to get the hidden variable X_h using Equation 3. X_h is used to represent a text description in the hidden layer.

$$X_h = \frac{\sum_{i=1}^{N} x_i}{N} \in \mathbb{R}^m \tag{3}$$

3.5 Output Layer

In the output layer, we use the softmax function (Bishop 2006) to compute a probability distribution over the predefined tags. When the number of tags is large, computing the linear classifier is expensive. The computational complexity is $O(km)$, where k is the number of tags and m is the dimension of the hidden layer. In order to improve the performance of FastTagRec, we use a hierarchical softmax function (Goodman 2001) based on Huffman coding tree (Mikolov et al 2013), which reduces the complexity to $O(m \log_2(k))$ during training.

The output layer corresponds to a binary tree. The leaf nodes denote the tags in a software information site. The weight of a leaf node represents the frequency of its tag. The number of the leaf nodes is k, and the number of non-leaf nodes is $(k-1)$. In the following we explain the notations used in our algorithm.

- p^t denotes the path from the root node to the leaf node corresponding to tag t.
- l^t denotes the number of nodes in the path p^t.
- $p_1^t, p_2^t, \ldots, p_{l^t}^t$ denote the l^t nodes in the path p^t. p_1^t is the root node and $p_{l^t}^t$ is the leaf node corresponding to tag t.
- $[c_2^t, c_3^t, \ldots, c_{l^t}^t](c_i^t \in \{0, 1\})$ denotes the Huffman coding of tag t. The coding consists of $l^t - 1$ bit. c_i^t denotes the encoding of i-th node in the path p^t. Root node does not have encoding.
- $\theta_1^t, \theta_2^t, \ldots, \theta_{l^t-1}^t(\theta_i^t \in \mathbb{R}^m)$ denote the vector of non-leaf nodes in the path p^t. θ_i^t denotes the vector of i-th non-leaf node in the path p^t.

Based on the Huffman tree, we first construct the probability distribution $p(t|X_h)(X_h \in \mathbb{R}^m)$ over tags. For a leaf node t, the number of branches in the path p^t is $l^t - 1$. Each branch in the path can be regarded as a binary classification process. For each non-leaf node, we need to specify the categories for the left and right children. Except the root node, each node in the Huffman tree corresponds to a Huffman code of 0 or 1. In this paper, a node with coding 0 is defined as a positive class, otherwise is defined as a negative class. The relationship is defined in Equation 4.

$$Category(p_i^t) = 1 - c_i^t, i = 2, 3, \ldots, l^t \tag{4}$$

Based on the softmax function, the probability that a node is classified as a positive class can be computed by Equation 5.

$$\sigma(X_h^\top \theta) = \frac{1}{1 + e^{-X_h^\top \theta}}, \tag{5}$$

where θ is a vector of node. The probability that a node is classified as a negative class is $1 - \sigma(X_h^\top \theta)$.

For each tag t in a software information site, there is a path p^t from the root to the leaf. In the Huffman tree. there are $l^t - 1$ branches in the path. Because each branch can be regarded as a binary classification process and each classification produces a probability of $p(c_i^t | X_h, \theta_{i-1}^t)$, we can get the probability of $p(t|X_h)$ by the product of the probabilities $p(c_i^t | X_h, \theta_{i-1}^t)(2 \leqslant i \leqslant l^t)$. Equation 6 describes the relationship.

$$p(t|X_h) = \prod_{i=2}^{l^t} p(c_i^t | X_h, \theta_{i-1}^t), \tag{6}$$

where $p(c_i^t | X_h, \theta_{i-1}^t)$ can be obtained by ~~Equation 7 or~~ Equation 7. If the Huffman code of the next node of the $i-1$-th node in the path p^t is 0 ($c_i^t = 0$), $p(c_i^t | X_h, \theta_{i-1}^t) = \sigma(X_h^\top \theta_{i-1}^t)$. If the next node of the $i-1$-th node in the path p^t is negative class ($c_i^t = 1$), $p(c_i^t | X_h, \theta_{i-1}^t) = 1 - \sigma(X_h^\top \theta_{i-1}^t)$.

~~$$p(c_i^t | X_h, \theta_{i-1}^t) = \begin{cases} \sigma(X_h^\top \theta_{i-1}^t), c_i^t = 0; \\ 1 - \sigma(X_h^\top \theta_{i-1}^t), c_i^t = 1; \end{cases} \tag{7}$$~~

$$p(c_i^t | X_h, \theta_{i-1}^t) = [\sigma(X_h^\top \theta_{i-1}^t)]^{1-c_i^t} \bullet [1 - \sigma(X_h^\top \theta_{i-1}^t)]^{c_i^t} \tag{7}$$

For each sample $o_i.d$ with tag: $o_i.t$, we maximize log-likelihood equation 8 over the tags. Equation 8 is a sub-item of equation 1. We can maximize equation 1 by maximizing equation 8 for each sample $(o_i.d, o_i.t)$ in S.

$$\zeta = \sum_{t \in \mathcal{TA}} \log p(t|o_i.d) = \sum_{t \in \mathcal{TA}} \log p(t|X_h) \tag{8}$$

Equation 8 can be converted to Equation 9 by Equation 6.

$$\zeta = \sum_{t \in \mathcal{TA}} \sum_{i=2}^{l^t} \{(1 - c_i^t) \bullet \log[\sigma(X_h^\top \theta_{i-1}^t)] + c_i^t \bullet \log[1 - \sigma(X_h^\top \theta_{i-1}^t)]\} \tag{9}$$

Equation 9 is the objective function of the model. We can maximize the object function by using stochastic gradient decent and a linearly decaying learning rate η. When a sample $o_i.d$ with tag $o_i.t$ is trained on the model, the training can be done by using stochastic gradient decent and a linearly decaying learning rate η on multiple CPUs simultaneously. We update all related parameters, including the shared parameters weight matrix A and the matrix $B = \{\theta_1^t, \theta_2^t, \ldots, \theta_{l^t-1}^t\}, (\theta_i^t \in \mathbb{R}^m)$.

The softmax function based on Huffman tree is efficient in searching for the most likely tag. Each node is associated with a probability of the path from the root node to itself. For the node t at depth l^t with parent nodes $p_1^t, p_2^t, \ldots, p_{l^t}^t$, its probability can be obtained by Equation 6. This means that the probability of a node is always lower than the one of its parents. Exploring the Huffman tree with a depth first search and tracking the maximum probability among the leaves allow us to discard any branch associated with a small probability. In practice, we observe a reduction of the complexity to $O(m \log_2(k))$. This method can be extended to compute the Top-K tags at the cost of $O(\log_2(k))$, using a binary heap.

4 Experiments and Results

In this section, we first evaluate the performance of our proposed FastTagRec automated tag recommender tool. All of these experiments were conducted on a 64-bit Intel Core i7 3.6G desktop computer with 64G RAM running Ubuntu 16.04. Then, we conduct a user study to evaluate the usefulness of our approach.

4.1 Benchmarks

We define a site as a large-scale site if the number of software objects in the site is more then 1 million, as a medium-scale site if the number of software objects in the site is between 100k to 1 million, and as a small-scale site if the number of software objects in the site is less than 100k. We have evaluated FastTagRec on one large-scale software information site `StackOverflow`, 3 medium-scale software information sites `Askubuntu`, `Serverfault`, `Unix` and 6 small-scale sites `Codereview`, `Freecode`, `Database Administrator`, `Wordpress`, `AskDifferent` and `Software Engineering`.

For the 3 medium-scale and 6 small-scale software information sites, we considered all of the software objects posted to the site before Dec 31st, 2016. For `StackOverflow`, we selected all of the software objects posted before July 1st, 2014, the same date set as in the prior work (Zhou et al 2017) to facilitate comparison in this empirical study. The code of FastTagRec and all of the data sets in our experiments described below can be accessed via the link `https://pan.baidu.com/s/1slujtU1`.

Before conducting experiments on these data sets we needed to remove rare tags and software objects. We define a tag to be rare if its number of appearances is less than or equal to a predefined threshold ts. A rare tag appears in a mature software information site under two scenarios. The first scenario is that the rare tag represents a rarely discussed topic. In this case, developers have yet to agree the tag is appropriate for the topic and thus they should be encouraged to create their own tags. The second scenario is that the tag is inappropriate, e.g. with spelling errors, for a popular topic. In either

scenario, rare tags should not be recommended. In the data pre-processing stage, we remove these rare tags from the software object. A software object is removed if all its tags are rare.

Table 1 summarizes the number of tags and software objects after removing the rare ones under different threshold values. We set the value of ts to 1, 50 and 10000 for `StackOverflow` and to 1 and 50 for 3 medium-scale and 6 small-scale software information sites. Threshold values ts 50 have been used in prior works (Xia et al 2013; Wang et al 2014; Zhou et al 2017).

For these software objects in Table 1, we further removed code snippets and screenshots from their descriptions. As code snippets are placed in specific HTML element components (<code>...<\code>). Code snippets can easily be removed through regular expression. That is, only the text in the description is preserved. It can be observed from Table 1 that the number of software objects ranges from about 40k to a quarter million for the small-scale software information sites and more than ten million for the large one.

For these sites `StackOverflow`, `Askubuntu`, `Serverfault`, `Unix`, `Codereview`, `Freecode`, `Database Administrator`, `Wordpress`, `AskDifferent` and `Software Engineering`, the average number of tags per software objects are 2.96, 2.69, 2.88, 2.78, 2.94, 3.48, 2.71, 2.43, 2.82 and 2.71 respectively.

It can be observed that many programmers prefer to give three tags for each posting. We also used a log function to linearly fit the frequency of tags and the number of tags with the same frequency. The schematic diagrams are depicted in Figures 5[a-j] respectively. The coefficient of the determination R^2 of the linear regression fitting are 0.6459, 0.666, 0.6751, 0.7153, 0.6459, 0.5666, 0.666, 0.5614, 0.6349 and 0.7262 respectively. The tags distribution in these data sets is a near power-law distribution. A similar conclusion was also obtained in related work by Beyer and Pinzger 2015.

In our experiments, we randomly selected 10,000 software objects and treated them as our test set V. The remaining software objects in the given software information site were used to recommend tags for the 10,000 selected ones. For each software object $o_i \in V$, we recommend k tags to form a tag set TR_i^k. We repeat the process ten times and compare FastTagRec against TagMulRec (Zhou et al 2017), a state-of-the-art tag recommendation method for these data sets. Because the other key related approach, EnTagRec is not scalable enough, as discussed in our related work section below, we were not able to use EnTagRec for comparison.

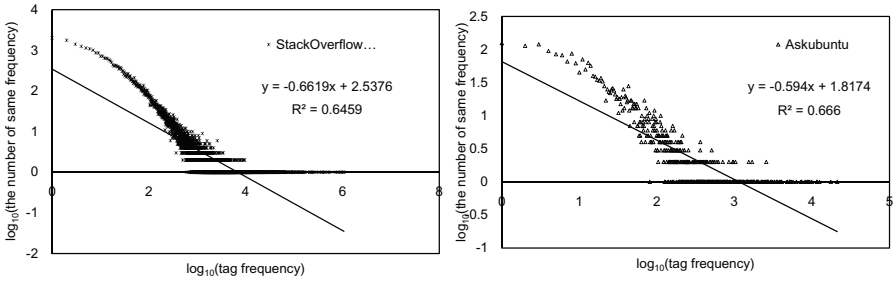
4.2 Evaluation Metrics

We use the same evaluation metrics as used for TagMulRec to facilitate comparison of the approaches. In the following we first define these evaluation metrics that are widely used in evaluating recommendation systems (Xia et al 2016a,b,c, 2015).

– Top-k prediction recall, denoted as *Recall@k*, is the percentage of top k recommend tags that are actually used by software objects. Given a

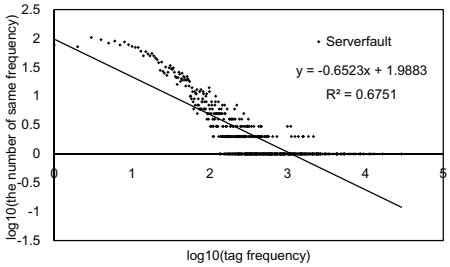
Table 1: Statistics of the nine datasets on different rare tag threshold values

Site Name	URL	*ts*	#software object	#tags
StackOverflow	http://www.stackoverflow.com	1	11203032	44265
		50	11193348	18952
		10000	10421906	427
Askubuntu	http://www.askubuntu.com	1	248630	3041
		50	246138	1146
Serverfault	http://www.serverfault.com	1	232996	3482
		50	231319	1312
Unix	http://www.unix.stackexchange.com	1	104744	2407
		50	103243	770
Codereview	http://www.codereview.stackexchange.com	1	39989	909
		50	39811	302
Freecode	http://www.github.com	1	47978	9018
		50	43644	274
Database Administrator	http://dba.stackexchange.com	1	51031	969
		50	50687	293
Wordpress	http://wordpress.stackexchange.com	1	71338	770
		50	70491	403
AskDifferent	http://apple.stackexchange.com	1	77978	1049
		50	77503	469
Software Engineering	http://softwareengineering.stackexchange.com	1	42782	1628
		50	41531	418

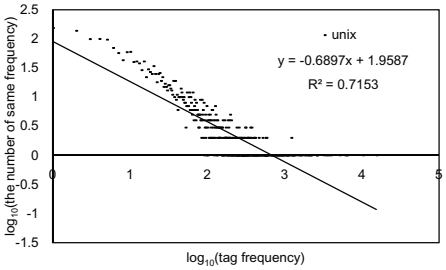


(a) Stackoverflow

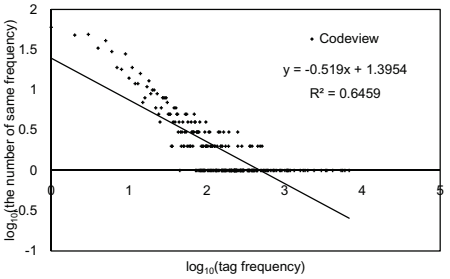
(b) Askubuntu



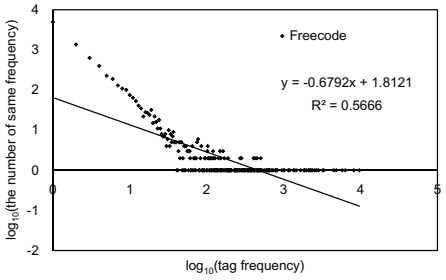
(c) Serverfault



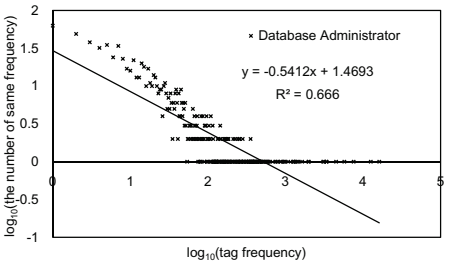
(d) Unix



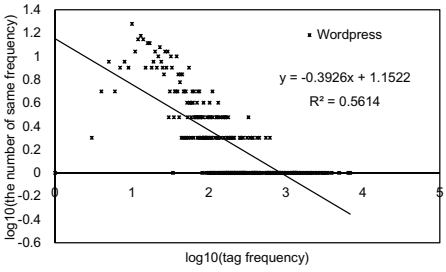
(e) Codereview



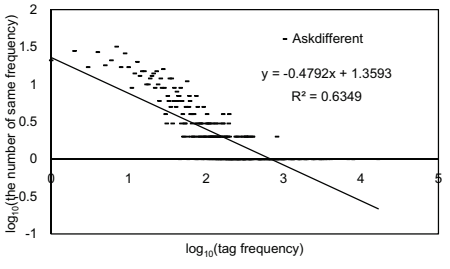
(f) Freecode



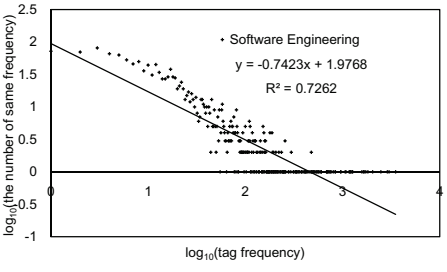
(g) DatabaseAdministrator



(h) Wordpress



(i) Askdifferent



(j) Software Engineering

Fig. 5: Distribution of the usage of tags (postcount) on these datasets.

software object o_i and its tags $o_i.T$, $Recall@k_i$ is computed by Equation 10, where TR_i^k is the top k tags recommended by FastTagRec for o_i. For example, if four out of top five recommended tags are actually used by a software object, the recall value is 80%. However, recall value favors small k. If only the first recommended tag is correct and all the rest are wrong, the recall value is still 100% if $k = 1$. This problem is addressed by the precision value below.

$$Recall@k_i = \begin{cases} \frac{|TR_i^k \cap o_i.T|}{k}, \mid o_i.T \mid > k. \\ \frac{|TR_i^k \cap o_i.T|}{|o_i.T|}, \mid o_i.T \mid \leq k. \end{cases} \tag{10}$$

Finally, given a set V of software objects , $Recall@k$ is defined by Equation 11.

$$Recall@k = \frac{\sum_{i=1}^{|V|} Recall@k_i}{\mid V \mid} \tag{11}$$

– Top-k prediction precision, denoted as $Precision@k$, is the percentage of the tags used by software objects that are among the top k recommended tags. TR_i^k. Given a software object o_i, $Precision@k_i$ for this particular software object is defined by Equation 12. The k value indicates the number of tags that we want to recommend to the developer. The $Precision@k_i$ is inversely proportional to the k value. For example, if a software object has two tags and they are both among the top five tags recommended to the developer (namely the k value is 5), the precision value is 40%. However, if the k value increases to 10, then the precision value will only be 20%. Clearly, a good precision rate in our work indicates a reasonable number of tags recommended to the developer. Note that if k is extremely large, the precision value can be very low. This is not true for recall.

$$Precision@k_i = \frac{\mid TR_i^k \cap o_i.T \mid}{k} \tag{12}$$

Given a set V of software objects, $Precision@k$ is computed by Equation 13.

$$Precision@k = \frac{\sum_{i=1}^{|V|} Precision@k_i}{\mid V \mid} \tag{13}$$

– Top-k Prediction F1-score, denoted as $F1\text{-}score@k$, combines Top-k prediction recall and Top-k prediction precision. Equation 14 gives the definition of $F1\text{-}score@k_i$ for software object o_i.

$$F1 - score@k_i = 2 \cdot \frac{Precision@k_i \cdot Recall@k_i}{Precision@k_i + Recall@k_i} \tag{14}$$

Given a set V of software objects, $F1\text{-}score@k$ is defined by Equation 15.

$$F1 - score@k = \frac{\sum_{i=1}^{|V|} F1 - score@k_i}{\mid V \mid} \tag{15}$$

We also compared the efficiency of the two methods. FastTagRec running time includes the training time and the prediction time. We use the average prediction time to evaluate the efficiency of FastTagRec as its training time is a one-off, off-line time cost. TagMulRec running time includes the time of constructing the candidate set and the time to recommend tags. Because TagMulRec constructs candidate sets dynamically, we use the average running time of tag recommendation to evaluate the efficiency of TagMulRec.

4.3 Experimental Results

In order to evaluate our FastTagRec and compare it with the state-of-the-art approach in a comprehensive way, we have conducted three groups of experiments to answer the following three questions.

1. RQ1: Compared with the existing state-of-the-art approach, how effective is FastTagRec?
2. RQ2: Compared with the existing state-of-the-art approach, how efficient is FastTagRec?
3. RQ3: Does the value of the rare tag threshold affect the performance of FastTagRec?

4.3.1 Results for RQ1

In this group of experiments we compare FastTagRec against TagMulRec on software information sites of different scales. Our experimental results are given in Table 2. In Table 2, Column 1 lists the names of the software information sites and Column 2 gives the threshold value 50. Tag threshold value ts 50 has been used in prior works (Xia et al 2013; Wang et al 2014; Zhou et al 2017). The rest of the columns compare FastTagRec with TagMulRec using the three metrics $Recall@k$, $Precision@k$ and $F1\text{-}score@k$. The top half and the bottom half of the table show the results when $k = 5$ and $k = 10$, respectively. The bold font in the table indicates the better results. The standard deviation is reported in brackets.

It can be observed that FastTagRec achieves better performance than TagMulRec in terms of $Recall@k$, $Precision@k$, $F1\text{-}score@k$ on all the k settings on one large-scale and three medium-scale software information sites.

For the six small-scale software information sites, FastTagRec achieves better performance than TagMulRec on 3 sites in terms of $Recall@5$, $Precision@5$, $F1\text{-}score@5$, and 4 sites in terms of $Recall@10$, $Precision@10$, $F1\text{-}score@10$. Wilcoxon signed-rank test (Song et al 2011; Zimmermann and Nagappan 2008) confirms that the performance improvement of FastTagRec is statistically significant (p-value < 0.001). It may not be enough to use the Wilcoxon test only to assess the significance of the differences of experimental results. Therefore, we use Cliff's Delta which is an effective measure for the magnitude of differences. Cliff's Delta (Macbeth et al 2011) indicates that the differences between the experimental results of our method and the baseline method are

significant (namely |d-value| is close to 1). In addition, as FastTagRec is based on neural network, the larger the scale of software information site, the better the effectiveness (Mikolov et al 2013).

4.3.2 Results for RQ2

In order to investigate the efficiency of FastTagRec, we compared FastTagRec against TagMulRec on different sized software information sites. The experimental results are given in Table 3. In Table 3, Column 1 lists the names of the software information sites and Column 2 gives the threshold values that are 1, 50 and 10000 (only for `StackOverflow`), and the third and fourth Column compare the training time needed to construct recommendation model. Although the training time is significant, it is a one-time expense. The last two columns compare the average time needed to recommend one tag.

It can be observed that the training time of FastTagRec is significantly shorter than TagMulRec on all the ts settings. FastTagRec achieves three orders of magnitude reduction for one large-scale software information site in prediction time on all the ts settings, indicating its capability in recommending large number of tags for larger scale software information sites. For three medium scale and six small scale software information sites, FastTagRec achieves one orders of magnitude reduction in prediction time on all the ts settings.

Therefore, we claim that based on these experimental results, FastTagRec is more efficient than TagMulRec for tag recommendation for various sized software information sites.

4.3.3 Results for RQ3

Both FastTagRec and TagMulRec remove rare tags. In this group of experiments, we investigated whether the rare tag threshold values affect the effectiveness and efficiency of FastTagRec. In order to do so, we evaluate *Recall*@k, *Precision*@k, F1-*score*@k (k=5 and 10) and time overhead of FastTagRec and TagMulRec under different tag threshold values.

Because prior works (Xia et al 2013; Wang et al 2014; Zhou et al 2017) limit the tag threshold value to 50, we used the tag threshold values of 1 and 50 for medium-scale and small-scale software information sites and to 1, 50 and 10,000 for large-scale software information sites. Table 4 shows the performance of FastTagRec and TagMulRec on tag threshold value 1.

Table 5 shows the performance of FastTagRec and TagMulRec on large-scale software information site `StackOverflow` with tag threshold values 1, 50, and 10,000. For the large-scale data set `Stackoverflow`, we noticed that when tag threshold value is 10,000, FastTagRec achieves higher *Recall*@k values than those with lower threshold values. However, the *Precision*@k and F1-*score*@k values are higher when threshold values are 50 and 1. When the rare tag threshold value is 50, FastTagRec achieves higher *Recall*@k values but

Table 2: The performance of FastTagRec and TagMulRec on tag threshold value 50

		Recall@5 (Standard Deviation)		Precision@5 (Standard Deviation)		F1−score@5 (Standard Deviation)	
Site Name	*ts*	TagMulRec	FastTagRec	TagMulRec	FastTagRec	TagMulRec	FastTagRec
StackOverflow	50	0.640 (0.002167)	**0.698** (0.002971)	0.343 (0.002164)	**0.386** (0.002261)	0.444 (0.002247)	**0.476** (0.002288)
Askubuntu	50	0.603 (0.002733)	**0.684** (0.002904)	0.271 (0.000814)	**0.346** (0.001584)	0.374 (0.001681)	**0.437** (0.002091)
Serverfault	50	0.622 (0.002445)	**0.666** (0.002654)	0.305 (0.001192)	**0.344** (0.001597)	0.403 (0.003040)	**0.435** (0.002998)
Unix	50	0.604 (0.001714)	**0.627** (0.002009)	0.294 (0.001632)	**0.309** (0.001567)	0.395 (0.002912)	**0.397** (0.002847)
Codereview	50	0.718 (0.001333)	**0.758** (0.001546)	0.377 (0.000618)	**0.398** (0.001132)	0.494 (0.000893)	**0.502** (0.001210)
Freecode	50	**0.659** (0.002403)	0.588 (0.002306)	**0.383** (0.001467)	0.343 (0.001106)	**0.485** (0.001546)	0.434 (0.001461)
Database Administrator	50	0.666 (0.000867)	**0.692** (0.001176)	0.313 (0.001887)	**0.332** (0.001562)	0.426 (0.002616)	**0.449** (0.001973)
Wordpress	50	0.605 (0.001695)	**0.632** (0.001673)	0.265 (0.000408)	**0.278** (0.000968)	0.368 (0.000616)	**0.386** (0.000871)
AskDifferent	50	**0.708** (0.002515)	0.689 (0.002316)	**0.372** (0.001094)	0.357 (0.001136)	**0.488** (0.002231)	0.471 (0.001890)
Software Engineering	50	**0.594** (0.001020)	0.582 (0.001205)	**0.253** (0.000262)	0.252 (0.000826)	**0.355** (0.000426)	0.352 (0.000813)
		Recall@10 (Standard Deviation)		Precision@10 (Standard Deviation)		F1−score@10 (Standard Deviation)	
Site Name	*ts*	TagMulRec	FastTagRec	TagMulRec	FastTagRec	TagMulRec	FastTagRec
StackOverflow	50	0.749 (0.001632)	**0.774** (0.001877)	0.205 (0.000798)	**0.217** (0.000894)	0.310 (0.001005)	**0.329** (0.001084)
Askubuntu	50	0.721 (0.002887)	**0.770** (0.002996)	0.166 (0.000497)	**0.198** (0.000562)	0.270 (0.001438)	**0.303** (0.001539)
Serverfault	50	0.716 (0.001254)	**0.753** (0.001792)	0.179 (0.001121)	**0.198** (0.001307)	0.287 (0.002206)	**0.304** (0.002201)
Unix	50	0.682 (0.002054)	**0.722** (0.002284)	0.169 (0.000235)	**0.182** (0.000973)	0.271 (0.001575)	**0.282** (0.001644)
Codereview	50	0.788 (0.001201)	**0.820** (0.001307)	0.211 (0.000619)	**0.218** (0.000874)	0.333 (0.000458)	**0.335** (0.000756)
Freecode	50	**0.758** (0.000976)	0.692 (0.001003)	**0.245** (0.000593)	0.219 (0.000731)	**0.364** (0.000998)	0.332 (0.000815)
Database Administrator	50	0.778 (0.001751)	**0.816** (0.001641)	0.188 (0.000336)	**0.201** (0.000498)	0.302 (0.000453)	**0.323** (0.000965)
Wordpress	50	0.725 (0.001193)	**0.765** (0.001264)	0.163 (0.000426)	**0.173** (0.000869)	0.266 (0.000412)	**0.283** (0.000649)
AskDifferent	50	**0.827** (0.002614)	0.815 (0.001895)	**0.222** (0.000834)	0.216 (0.000985)	**0.350** (0.001318)	0.342 (0.000993)
Software Engineering	50	0.704 (0.001706)	**0.708** (0.001509)	0.153 (0.000459)	**0.157** (0.000870)	0.252 (0.000739)	**0.257** (0.001147)

Table 3: The time efficiency of FastTagRec and TagMulRec

		Training Time(s)		Prediction Time(ms)	
Site Name	ts	TagMulRec	FastTagRec	TagMulRec	FastTagRec
StackOverflow	1	43201	**29569**	490	**0.354**
StackOverflow	50	41808	**28783**	162	**0.354**
StackOverflow	10000	39087	**1806**	160	**0.216**
Askubuntu	1	997	**765**	0.055	**0.00317**
Askubuntu	50	908	**372**	0.023	**0.00196**
Serverfault	1	430	**385**	0.043	**0.00351**
Serverfault	50	246	**87**	0.025	**0.00216**
Unix	1	960	**110**	0.096	**0.00266**
Unix	50	621	**101**	0.062	**0.00166**
Codereview	1	689	**122**	0.068	**0.00223**
Codereview	50	495	**83**	0.049	**0.00189**
Freecode	1	143	**79**	0.045	**0.00683**
Freecode	50	95	**48**	0.042	**0.00310**
Database Administrator	1	374	**59**	0.038	**0.00241**
Database Administrator	50	311	**32**	0.031	**0.00193**
Wordpress	1	486	**63**	0.048	**0.00243**
Wordpress	50	361	**34**	0.036	**0.00241**
AskDifferent	1	671	**184**	0.024	**0.00216**
AskDifferent	50	211	**90**	0.016	**0.00214**
Software Engineering	1	654	**79**	0.022	**0.00233**
Software Engineering	50	205	**44**	0.015	**0.00231**

achieves lower *Precision*@k and $F1$-*score*@k when tag threshold is 1, except *Precision*@5.

For the large-scale software information site with different tag threshold values, the prediction time of FastTagRec is the same order of magnitude. For the 3 medium-scale and 6 small-scale software information sites, when tag threshold value is 50, FastTagRec achieves higher *Recall*@k and *Precision*@k values than tag threshold 1, except for very few cases. In summary, the difference on tag threshold values has very limited effect on the $F1$-*score*@k values and time efficiency of the two compared approaches in these experiments.

4.4 User Study

To determine whether FastTagRec is effective in practice, we have designed and run a user study with five developers including three M.S. and two Ph.D. students. The five developers stated that they had moderate to high expertise in Python/Java programming/debugging and everyone has experience of at least two years. The five developers were asked to complete two tasks on their own time and no incentives were provided to these developers to complete these tasks.

We first selected 100 postings from `StackOverflow`. Among these postings, there are 50 postings about Python programming/debugging, and the other 50 postings about Java programming/debugging. The average number of tags per posting is 3. We use FastTagRec to recommend 10 tags for each posting. Five developers were asked to compare the recommended tags with the ground truth over 100 postings, and select any other appropriate tags in the recommended tags in addition to the ground truth for each posting. For a given posting, if more than three developers choose a specific tag from these recommended tags in addition to the ground truth, we think the tags is a good supplement to tag the given posting. We collected all responses from the first task. The results show that more than 80% postings received at least one supplement tags (21% postings received one supplement tag, 43% postings received two supplement tags and 17% postings received more than three supplement tags). The first task confirms that FastTagRec can provide useful supplement tags for developers.

Next, we selected 150 popular projects from `GitHub` and collected the readme files of these projects which are text files that often include project's description and instruction. As tagged projects in `GitHub` are rarely tagged, rich tags in `StackOverflow` can be used to tag these projects. We asked five developers to read these readme files and tag these projects using FastTagRec tool which is trained based on `StackOverflow` data set. Five developers were also asked to rate the level of difficulty in using FastTagRec to tag projects as easy, moderate, or difficult. Similar to the first task, for a given project, if more than three developers choose a specific tag from these recommended tags, we think the tag is appropriate to tag the given project. We collected all responses from the task. The results shows that each project received an average of 3.6

Table 4: The performance of FastTagRec and TagMulRec on tag threshold value 1

		Recall@5 (Standard Deviation)		Precision@5 (Standard Deviation)		F1−score@5 (Standard Deviation)	
Site Name	*ts*	TagMulRec	FastTagRec	TagMulRec	FastTagRec	TagMulRec	FastTagRec
StackOverflow	1	0.628 (0.002201)	**0.692** (0.002839)	0.340 (0.002187)	**0.385** (0.002249)	0.441 (0.002238)	**0.492** (0.002398)
Askubuntu	1	0.592 (0.000821)	**0.654** (0.002562)	0.275 (0.001129)	**0.340** (0.001673)	0.376 (0.001498)	**0.425** (0.001905)
Serverfault	1	0.552 (0.001808)	**0.623** (0.002242)	0.282 (0.002042)	**0.338** (0.002183)	0.373 (0.002685)	**0.419** (0.002876)
Unix	1	0.535 (0.002207)	**0.585** (0.002306)	0.274 (0.000418)	**0.307** (0.001230)	0.362 (0.002397)	**0.386** (0.002548)
Codereview	1	0.673 (0.001568)	**0.709** (0.001604)	0.381 (0.001704)	**0.395** (0.001875)	0.486 (0.001513)	**0.489** (0.001782)
Freecode	1	0.477 (0.002710)	**0.508** (0.002837)	0.304 (0.001394)	**0.308** (0.001452)	0.371 (0.001205)	**0.383** (0.001293)
Database Administrator	1	0.628 (0.000925)	**0.651** (0.001243)	0.318 (0.000598)	**0.330** (0.001085)	0.422 (0.000992)	**0.438** (0.001138)
Wordpress	1	0.587 (0.000578)	**0.604** (0.000786)	0.266 (0.000431)	**0.275** (0.000875)	0.365 (0.000755)	**0.377** (0.000927)
AskDifferent	1	**0.687** (0.001224)	0.663 (0.001357)	**0.376** (0.000501)	0.358 (0.001013)	**0.486** (0.000679)	0.465 (0.001764)
Software Engineering	1	**0.521** (0.000727)	0.491 (0.001018)	**0.251** (0.000469)	0.246 (0.000795)	**0.339** (0.000518)	0.327 (0.000971)

		Recall@10 (Standard Deviation)		Precision@10 (Standard Deviation)		F1−score@10 (Standard Deviation)	
Site Name	*ts*	TagMulRec	FastTagRec	TagMulRec	FastTagRec	TagMulRec	FastTagRec
StackOverflow	1	0.703 (0.001583)	**0.770** (0.001793)	0.194 (0.000763)	**0.218** (0.000872)	0.304 (0.000997)	**0.335** (0.001052)
Askubuntu	1	0.681 (0.002807)	**0.737** (0.002912)	0.162 (0.000687)	**0.194** (0.000837)	0.261 (0.001338)	**0.296** (0.001407)
Serverfault	1	0.675 (0.001216)	**0.708** (0.001685)	0.176 (0.001368)	**0.196** (0.001432)	0.280 (0.001842)	**0.297** (0.001935)
Unix	1	0.625 (0.002363)	**0.665** (0.002479)	0.163 (0.000651)	**0.178** (0.000891)	0.259 (0.001879)	**0.272** (0.001952)
Codereview	1	0.741 (0.000911)	**0.771** (0.001153)	0.213 (0.000267)	**0.217** (0.000567)	**0.332** (0.000294)	0.331 (0.000988)
Freecode	1	0.552 (0.000623)	**0.598** (0.001047)	0.194 (0.000188)	**0.199** (0.000703)	0.287 (0.000303)	**0.298** (0.000428)
Database Administrator	1	0.735 (0.001463)	**0.767** (0.001536)	0.191 (0.000392)	**0.200** (0.000706)	0.303 (0.000476)	**0.317** (0.000862)
Wordpress	1	0.707 (0.001707)	**0.732** (0.001870)	0.164 (0.000274)	**0.171** (0.000563)	0.267 (0.000440)	**0.277** (0.000657)
AskDifferent	1	**0.807** (0.001238)	0.790 (0.001010)	**0.225** (0.000346)	0.218 (0.000769)	**0.352** (0.000389)	0.342 (0.000672)
Software Engineering	1	**0.617** (0.001345)	0.603 (0.000984)	0.152 (0.000473)	**0.155** (0.000829)	0.244 (0.000728)	**0.246** (0.001106)

Table 5: FastTagRec VS TagMulRec on `StackOverflow` with tag threshold values 1, 50, 10000

		Recall@5 (Standard Deviation)		Precision@5 (Standard Deviation)		F1−score@5 (Standard Deviation)	
Site Name	ts	TagMulRec	FastTagRec	TagMulRec	FastTagRec	TagMulRec	FastTagRec
`StackOverflow`	1	0.628 (0.002201)	**0.692** (0.002839)	0.340 (0.002187)	**0.385** (0.002249)	0.441 (0.002238)	**0.492** (0.002398)
`StackOverflow`	50	0.640 (0.002167)	**0.698** (0.002971)	0.343 (0.002164)	**0.386** (0.002261)	0.444 (0.002247)	**0.476** (0.002288)
`StackOverflow`	10000	0.809 (0.003458)	**0.870** (0.003742)	0.310 (0.002101)	**0.349** (0.002136)	0.449 (0.002302)	**0.476** (0.002329)
		Recall@10 (Standard Deviation)		Precision@10 (Standard Deviation)		F1−score@10 (Standard Deviation)	
Site Name	ts	TagMulRec	FastTagRec	TagMulRec	FastTagRec	TagMulRec	FastTagRec
`StackOverflow`	1	0.703 (0.001583)	**0.770** (0.001793)	0.194 (0.000763)	**0.218** (0.000872)	0.304 (0.000997)	**0.335** (0.001052)
`StackOverflow`	50	0.749 (0.001632)	**0.774** (0.001877)	0.205 (0.000798)	**0.217** (0.000894)	0.310 (0.001005)	**0.329** (0.001084)
`StackOverflow`	10000	0.892 (0.002582)	**0.919** (0.002633)	0.176 (0.000903)	**0.187** (0.000943)	0.294 (0.001193)	**0.301** (0.001256)

appropriate tags. Five developers unanimously agreed that FastTagRec is very useful and easy to use. The second task confirms that FastTagRec is a useful tool for tagging software projects.

5 Discussion

In this section, we will first share some of the important lessons that we learned in implementing the work in this paper, followed by the discussion on threats to Validity.

5.1 Implications on Our Work

While experimental results affirm that our method is suitable for tag recommendation in software information sites, how to reduce the training and prediction time also needs attention. To this end, we identified some implications which we take from implementing our method.

1. *Performance Improvement over Baseline Method.* As discussed in the related work, the baseline method only utilizes a small portion of software information sites that is most relevant to a given software object. To improve performance, our method utilizes shared parameters among features and tags to utilize all information in software information sites. To further improve performance, our method uses a bag of n-grams as additional features to capture partial information about the local word order (the rank constraint of word).
2. *Costs Reduction over Baseline Method.* It is important to consider the training and prediction costs of a method before applying it. Three tricks are used in our method to reduce these costs. Specifically, first, the network structure of our method is very simple, only with three layers: input layer, hidden layer and output layer. Second, the hash trick introduced in the input layer can help quickly find feature vector and use less memory. Third, the hierarchical softmax trick in the output layer based on Huffman coding tree reduces the complexity of our model during the training phase.
3. *Implication for Researchers and Practitioners.* The major motivation for researchers and practitioners is to use less cost to achieve better results. Our work makes a useful attempt to explore better method for our research question using the neural networks technology.

5.2 Threats to Validity

There are several threats that can potentially affect the validity of our results.

1. *Potential Experimental Errors.* Threats to internal validity relates to errors in our experiments. The authors have carefully checked the experiments and data sets, but there still could be experimental errors in the set up that that we did not notice.

2. *Potentially Biased Results.* Our tag recommendation assumes that existing tags in a software information site are correct. However, human errors are inevitable. We do apply some filtering rules, such as time interval of data set, to alleviate the problem. These filtering rules have also been used in other past research (Xia et al 2013; Wang et al 2014; Zhou et al 2017). However, this issue, such as how to deal with a large number of synonymous tags, cannot be completely solved. In the user study, these developers may be a potential subjective bias when select tags to label selected postings or projects. To mitigate potential subjective bias, a specific tag is considered as a good choice or supplement for a given object only when more than three developers haven chosen it.
3. *Generalizability of Algorithms.* External threats to validity relate to how generalisable experimental results can be. In this research, we have evaluated FastTagRec on one large scale, three medium-scale and six small scale software information sites. There are more than 11 million software objects in the large scale software information site. Even so, more case studies are needed to generalize our findings to other sites and kinds of software objects and their tags. In the future, more software information sites will be used to further evaluate FastTagRec.
4. *Suitability of Evaluation Metrics.* In this paper, *Recall*@k, *Precision*@k and $F1$-*score*@k are used as our evaluation metrics. *Recall*@k and *Precision*@k have been used in past research to evaluate the performance of tag recommendation for software information sites (Xia et al 2013; Wang et al 2014; Al-Kofahi et al 2010) and for social media and network (Zangerle et al 2011; Wang et al 2013a; Yang et al 2015, 2014). It is possible that more suitable metrics can be adopted. For example, since our tag recommendation is a multi-classification process (Cai et al 2011), the evaluation metrics of multi-label classification approaches (Tsoumakas and Katakis 2006; Zhang and Zhou 2007) will be used in our future work.
5. *Model Scalability.* In this paper, we only utilized the textual content of the description of software objects. We removed all code snippets and screenshots form the descriptions. We will attempt to utilize these code snippets and extracted information from screenshots in the description to extend our model in our future work.

6 Related Work

Tag recommendation has been a hot research problem in the fields of social network and data mining for some time (Sigurbjörnsson and Van Zwol 2008; Rendle and Schmidt-Thieme 2010; Yin et al 2010; Wang et al 2013b; Jäschke et al 2007). Automatic tag recommendation in software engineering was first proposed by Al-Kofahi et. al. in 2010 (Al-Kofahi et al 2010). Al-Kofahi et al. proposed a method called TAGREC to automatically recommend tags for work items in IBM Jazz. TAGREC was based on the fuzzy set theory and considered the dynamic evolution of a system. Later a method called TAG-

COMBINE (Xia et al 2013) was proposed to automatically recommend tags for software objects in software information sites. It consists of three components: a multi-label ranking component, a similarity based ranking component, and a tag-term based ranking component. The multi-label ranking approach adopted by TAGCOMBINE limits its application to relatively small datasets. For a large-scale software information site such as `StackOverflow`, TAGCOMBINE has to train more than forty thousand binary classifier models and the size of each training set is more than ten million. A more recent approach called EnTagRec (Wang et al 2014) outperforms TAGCOMBINE in terms of *Recall* and *Precision* metrics. EnTagRec consists of two components: Bayesian inference component and Frequentist inference component. However, EnTagRec is not scalable as well, as it also utilizes all information in software information sites to recommend tags for a software object. Lately, a state-of-the-art tags recommendation method TagMulRec is proposed by Zhou et. al. in 2017 (Zhou et al 2017). For a given software object, TagMulRec prunes the large-scale categories (tags) into a much smaller set of target category candidates for similarity distance computation. In addition, neither TAGCOMBINE nor EnTagRec adapts to the dynamic evolution of software information sites. In contrast, TagMulRec is scalable and is able to handle continuous updates in the software information sites. However, TagMulRec only utilizes a small portion of software information sites that is most relevant to a given software object. Our method FastTagRec utilizes shared parameters among features and tags to utilize all information in software information sites and avoid the limitation of generalization in the context of large output space where some tags have very few examples.

In the field of software engineering, tags have become widely used for information finding, team co-ordination and co-operation, and helping to bridge socio-technical issues (Storey et al 2010; Begel et al 2010; Treude and Storey 2009; Thung et al 2012; Wang et al 2012; Beyer and Pinzger 2015, 2016). Storey et. al. proposed a set of pertinent research questions (Storey et al 2010), which strives to understand the benefits, risks and limitations of using social media in software development at the team, project and community level, around community involvement, project coordination, project management and individual software development activities. Begel et al. described the potential benefits (Begel et al 2010) for social media to both improve communication and coordination in software development teams and support of the creation of new kinds of software development communities. Treude et al. explored how tagging is used to bridge the gap between technical and social aspects of managing work items (Treude and Storey 2009). They conducted an empirical study on how tagging has been adopted and adapted over the two year of a large project with 175 developers. Their results showed that the tagging mechanism had become a significant part for many informal processes (Treude and Storey 2009). Thung et al. detected similar software application using software tags Thung et al (2012). Wang et al. analyzed tags of projects in FREECODE to infer semantic relationships among the tags, and express the relationships as a taxonomy (Wang et al 2012. Beyer et al. designed a tag synonym sug-

gestion tool TSST to alleviates tag synonyms issue in `StackOverflow` (Beyer and Pinzger 2015, 2016).

Our primary related work is the work by Mikolov et al.(2013) (Mikolov et al 2013), who proposed a CBOW model to get word vectors in the NLP task. The CBOW model is based on artificial neural network techniques. An artificial neural network makes the information flow from input level through hidden levels to output level along connections with adjustable weights. The architecture of deep neural networks contains many hidden levels. Our method only contains a single hidden layer, which ensures that FastTagRec can work quickly in large-scale software information sites. Deep neural networks have been utilized in other software engineering tasks, such as code clone detection (White et al 2016), mining software repositories (White et al 2015; Xu et al 2016; Gu et al 2016), etc.

7 Conclusion and Future Work

In this paper, we presented a scalable tag recommendation method called FastTagRec for software information sites. FastTagRec achieves accuracy and efficiency by (1) constructing an suitable framework based on single-hidden layer neural networks, (2) exploiting the rank constraint of word, (3) utilizing shared parameters among features and avoiding the limitation in the context of large tag output space. We implemented FastTagRec and evaluated its performance on ten software information sites with large number of software objects and tags. The evaluation was conducted by recommending tags for randomly selected 10,000 software objects in each software information site. The experimental results confirmed that FastTagRec is much more effective and efficient than the stat-of-the-art approach. In order to evaluate FastTagRec in more realistic settings, we first ask five developers to compare the recommended tags with the ground truth over postings. Then, five developers are asked to tag open source projects and rate the usefulness of FastTagRec. The user study confirms that FastTagRec is useful for developers in the real scene.

Our current work is based on text only. In the future, we plan to consider code snippets and screenshots to make our tag recommendation more accurate. We will also conduct experiments on more large-scale software information sites with more evaluation metrics. The proposed method in this paper provides a framework that can potentially be used to solve other problems in software engineering, such as bug triage and code-reviewer recommendation. In the future, we will explore new applications for the framework.

References

Al-Kofahi JM, Tamrawi A, Nguyen TT, Nguyen HA, Nguyen TN (2010) Fuzzy set approach for automatic tagging in evolving software. In: International Conference on Software Maintenance, IEEE, pp 1–10

Begel A, DeLine R, Zimmermann T (2010) Social media for software engineering. In: Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM, pp 33–38

Behley J, Steinhage V, Cremers AB (2013) Laser-based segment classification using a mixture of bag-of-words. In: International Conference on Intelligent Robots and Systems, IEEE, pp 4195–4200

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: IEEE/ACM 13th Working Conference on Mining Software Repositories, IEEE, pp 430–440

Bishop CM (2006) Pattern recognition. Machine Learning 128:1–58

Cai L, Zhou G, Liu K, Zhao J (2011) Large-scale question classification in cqa by leveraging wikipedia semantic knowledge. In: Proceedings of the 20th ACM international conference on Information and knowledge management, ACM, pp 1321–1330

Fowkes J, Sutton C (2016) Parameter-free probabilistic api mining across github. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 254–265

Goodman J (2001) Classes for fast maximum entropy training. In: International Conference on Acoustics, Speech, and Signal Processing, IEEE, vol 1, pp 561–564

Gu X, Zhang H, Zhang D, Kim S (2016) Deep api learning. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 631–642

Hindle A, Alipour A, Stroulia E (2016) A contextual approach towards more accurate duplicate bug report detection and ranking. Empirical Software Engineering 21(2):368–410

Hou D, Mo L (2013) Content categorization of api discussions. In: International Conference on Software Maintenance, IEEE, pp 60–69

Jäschke R, Marinho L, Hotho A, Schmidt-Thieme L, Stumme G (2007) Tag recommendations in folksonomies. In: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, pp 506–514

Macbeth G, Razumiejczyk E, Ledesma RD (2011) Cliffs delta calculator: A non-parametric effect size program for two groups of observations. Universitas Psychologica 10(2):545–555

Michaud HM, Guarnera DT, Collard ML, Maletic JI (2016) Recovering commit branch of origin from github repositories. In: International Conference on Software Maintenance and Evolution, IEEE, pp 290–300

Mikolov T, Deoras A, Povey D, Burget L, Černockỳ J (2011) Strategies for training large scale neural network language models. In: IEEE Workshop on Automatic Speech Recognition and Understanding, IEEE, pp 196–201

Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:13013781

Rendle S, Schmidt-Thieme L (2010) Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the third ACM international conference on Web search and data mining, ACM, pp 81–90

Robillard MP, Medvidović N (2016) Disseminating architectural knowledge on open-source projects: a case study of the book architecture of open-source applications. In: Proceedings of the 38th International Conference on Software Engineering, ACM, pp 476–487

Sigurbjörnsson B, Van Zwol R (2008) Flickr tag recommendation based on collective knowledge. In: Proceedings of the 17th international conference on World Wide Web, ACM, pp 327–336

Song Q, Jia Z, Shepperd M, Ying S, Liu J (2011) A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering 37(3):356–370

Storey MA, Treude C, van Deursen A, Cheng LT (2010) The impact of social media on software engineering practices and tools. In: Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM, pp 359–364

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

Thung F, Kochhar PS, Lo D (2014) Dupfinder: integrated tool support for duplicate bug report detection. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, ACM, pp 871–874

Treude C, Robillard MP (2016) Augmenting api documentation with insights from stack overflow. In: Proceedings of the 38th International Conference on Software Engineering, ACM, pp 392–403

Treude C, Storey MA (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

Tsoumakas G, Katakis I (2006) Multi-label classification: An overview. Dept of Informatics, Aristotle University of Thessaloniki, Greece

Wang H, Chen B, Li WJ (2013a) Collaborative topic regression with social regularization for tag recommendation. In: International Joint Conference on Artificial Intelligence, ACM, pp 2719–2725

Wang Q, Ruan L, Zhang Z, Si L (2013b) Learning compact hashing codes for efficient tag completion and prediction. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management, ACM, pp 1789–1794

Wang S, Manning CD (2012) Baselines and bigrams: Simple, good sentiment and topic classification. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, Association for Computational Linguistics, pp 90–94

Wang S, Lo D, Jiang L (2012) Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In: International

Conference on Software Maintenance, IEEE, pp 604–607
Wang S, Lo D, Vasilescu B, Serebrenik A (2014) Entagrec: An enhanced tag recommendation system for software information sites. In: ICSME, pp 291–300
White M, Vendome C, Linares-Vásquez M, Poshyvanyk D (2015) Toward deep learning software repositories. In: IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE, pp 334–345
White M, Tufano M, Vendome C, Poshyvanyk D (2016) Deep learning code fragments for code clone detection. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, pp 87–98
Xia X, Lo D, Wang X, Zhou B (2013) Tag recommendation in software information sites. In: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, pp 287–296
Xia X, Feng Y, Lo D, Chen Z, Wang X (2014) Towards more accurate multi-label software behavior learning. In: IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering, IEEE, pp 134–143
Xia X, Lo D, Wang X, Zhou B (2015) Dual analysis for recommending developers to resolve bugs. J Softw Evol Process 27(3):195–220
Xia X, Lo D, Ding Y, Al-Kofahi JM, Nguyen TN, Wang X (2016a) Improving automated bug triaging with specialized topic model. IEEE Transactions on Software Engineering Accpted
Xia X, Lo D, Pan SJ, Nagappan N, Wang X (2016b) Hydra: Massively compositional model for cross-project defect prediction. IEEE Transactions on Software Engineering Accpted
Xia X, Lo D, Wang X, Yang X (2016c) Collective personalized change classification with multi-objective search. IEEE Transactions on Reliability
Xu B, Ye D, Xing Z, Xia X, Chen G, Li S (2016) Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, pp 51–62
Yang D, Xiao Y, Song Y, Zhang J, Zhang K, Wang W (2014) Tag propagation based recommendation across diverse social media. In: Proceedings of the 23rd International Conference on World Wide Web, ACM, pp 407–408
Yang D, Xiao Y, Tong H, Zhang J, Wang W (2015) An integrated tag recommendation algorithm towards weibo user profiling. In: International Conference on Database Systems for Advanced Applications, Springer, pp 353–373
Yang L, Qiu M, Gottipati S, Zhu F, Jiang J, Sun H, Chen Z (2013) Cqarank: jointly model topics and expertise in community question answering. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management, ACM, pp 99–108
Yin D, Xue Z, Hong L, Davison BD (2010) A probabilistic model for personalized tag prediction. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 959–968
Zangerle E, Gassler W, Specht G (2011) Using tag recommendations to homogenize folksonomies in microblogging environments. In: International Confer-

ence on Social Informatics, Springer, pp 113–126
Zhang ML, Zhou ZH (2007) Ml-knn: A lazy learning approach to multi-label learning. Pattern recognition 40(7):2038–2048
Zhao Z, Zhang L, He X, Ng W (2015) Expert finding for question answering via graph regularized matrix completion. IEEE Transactions on Knowledge and Data Engineering 27(4):993–1004
Zhou P, Liu J, Yang Z, Zhou G (2017) Scalable tag recommendation for software information sites. In: The 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)
Zimmermann T, Nagappan N (2008) Predicting defects using network analysis on dependency graphs. In: International Conference on Software Engineering, IEEE, pp 531–540