Integrating Multi-level Tag Recommendation with External Knowledge Bases for Automatic Question Answering

EDUARDO LIMA, WEISHI SHI, XUMIN LIU, and QI YU, Rochester Institute of Technology

We focus on using natural language unstructured textual Knowledge Bases (KBs) to answer questions from community-based Question-and-Answer (Q&A) websites. We propose a novel framework that integrates multi-level tag recommendation with external KBs to retrieve the most relevant KB articles to answer user posted questions. Different from many existing efforts that primarily rely on the Q&A sites' own historical data (e.g., user answers), retrieving answers from authoritative external KBs (e.g., online programming documentation repositories) has the potential to provide rich information to help users better understand the problem, acquire the knowledge, and hence avoid asking similar questions in future. The proposed multi-level tag recommendation best leverages the rich tag information by first categorizing them into different semantic levels based on their usage frequencies. A post-tag co-clustering model, augmented by a two-step tag recommender, is used to predict tags at different levels for a given user posted question. A KB article retrieval component leverages the recommended multi-level tags to select the appropriate KBs and search/rank the matching articles thereof. We conduct extensive experiments using real-world data from a Q&A site and multiple external KBs to demonstrate the effectiveness of the proposed question-answering framework.

CCS Concepts: • **Information systems** → **Social tagging systems**; **Clustering**;

Additional Key Words and Phrases: Question answering, tag recommendation, co-clustering

ACM Reference format:
Eduardo Lima, Weishi Shi, Xumin Liu, and Qi Yu. 2019. Integrating Multi-level Tag Recommendation with External Knowledge Bases for Automatic Question Answering. *ACM Trans. Internet Technol.* 19, 3, Article 34 (May 2019), 22 pages.
https://doi.org/10.1145/3319528

1 INTRODUCTION

A Question-and-Answer (Q&A) site is an online community that builds its user base by providing means for users to have their questions answered. The importance of this kind of system cannot be overstated, as we have seen leading IT companies in the past decade have grown large communities around this framework. For example, Yahoo Answers, one of the pioneers, currently has an estimated 4.6M unique U.S. visitors per month; Quora, a more recent one, has an estimated 1.9M

This research was supported in part an NSF IIS Award No. IIS-1814450 and an ONR Award No. N00014-18-1-2875. Lima is also supported by CAPES (Brazil's Federal Agency for Support and Evaluation of Graduate Education), CNPq (Brazil's Council for Scientific and Technological Development), and IFRN (Brazil's Federal Institute of Rio Grande do Norte).
Authors' addresses: E. Lima, W. Shi, X. Liu, and Q. Yu, Rochester Institute of Technology, 152 Lomb Memorial Drive, Rochester, NY 14623-5608; emails: {ecl7037, ws7586, xumin.liu, qi.yu}@rit.edu.

1533-5399/2019/05-ART34 $15.00
https://doi.org/10.1145/3319528

unique U.S. visitors per month; and Stack Overflow has the most impressing numbers, with 12.5M U.S. unique visitors and 48.3M global unique visitors[1] per month.

Even though there is a great number of answers being submitted by users, one persistent problem is still observed: *It usually takes hours or days before a question gets acceptable answers. Some questions have not been even answered at all.* For users, it could be frustrating if they do not receive appropriate answers for their questions or need to wait for an undetermined long period before getting answers. To deal with this situation, many existing systems, such as StackOverflow, support users to leverage the historical data collected by the system to find answers. In particular, standard information retrieval techniques, such as vector space model, tf-idf weighting, and ranked retrieval, can be employed to support keyword-based search over existing questions/answers. If similar questions with satisfactory answers already provided can be successfully located, then users may directly exploit these answers as solutions to their questions.

Another interesting strategy that could serve as an effective alternative or complement the above automatic question-answering approach is to extend the Q&A sites with external information sources to provide possible answers. Besides suggesting answers from similar posts, information retrieved from authoritative knowledge sources (e.g., online Java documentations for questions related to Java programming) could be very helpful. Following this line, in this article, we aim to discover information from external data sources that provides users with the content that fits their information need. Meanwhile, it can offer additional details and context to motivate users to engage in learning the knowledge while trying to find answers. Figure 1 shows an example, where the user-provided answer refers to the official reference. The reference has a rich explanation that not only answers the question but also enables the user to further understand related concepts and also some reasoning about the matter.

In this article, we propose to effectively leverage information from external authoritative data sources, referred to as *Knowledge Bases or KBs*, to automate question answering and promote user learning. We focus on software development-related Q&A communities, as they so far have attracted the largest number of users, but the proposed approach can be readily applied to other domains as well. After going over multiple Q&A communities, it became clear that online programming documentation libraries (e.g., Java and Python API references) are good candidates to act as appropriate KBs to answer user questions from these communities. Figure 2 shows another example where we can see a user provided answer links to the official reference contents.

Knowledge retrieval from external KBs for question answering has drawn significant attentions in recent years [4, 5, 11, 17, 20, 23, 24, 26–28]. However, most existing efforts focused on structured questions and/or KBs, such as relational databases and ontologies, leveraging the structural information to perform queries on KBs. This makes them not applicable to community Q&A websites due to the free-form and unstructured nature of both posts and KBs. Our approach, in contrast, focuses on using natural language unstructured textual KBs, which is more commonly available than structured ones, to find answers for user questions. The research challenges that we tackle include: (1) When considering multiple KBs, it is important to identify the proper one for a question, such as choosing Java instead of PhP API documentation for a Java related question. (2) Within a KB, it is nontrivial to discover the semantically relevant articles in a KB for a question, due to the free text forms of both question and answers. (3) The tremendous amount of data from both the Q&A sites and the external KBs poses scalability issue, which leads to significant computational overhead if using traditional text mining techniques.

[1] https://www.quantcast.com/.

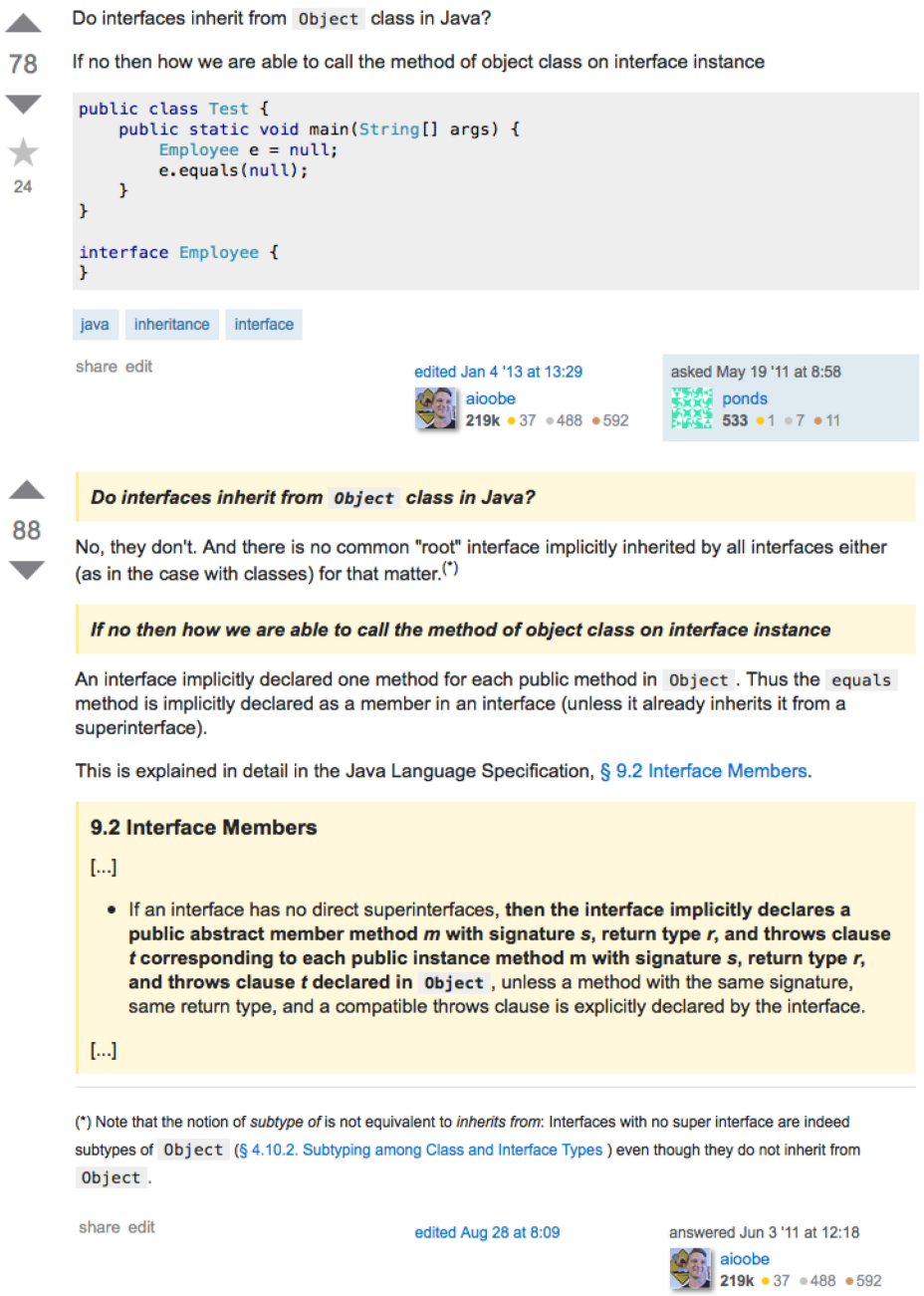


Fig. 1. A question on Java programming and its corresponding answer.

We propose a novel framework that integrates multi-level tag recommendation with multiple external KBs to address the challenges as outlined above. Our key contributions are summarized as follows:

- We develop a **Multi-level Tag Recommendation** (MLTR) model to best leverage the rich tag information available in the Q&A sites. We categorize a large collection of tags into

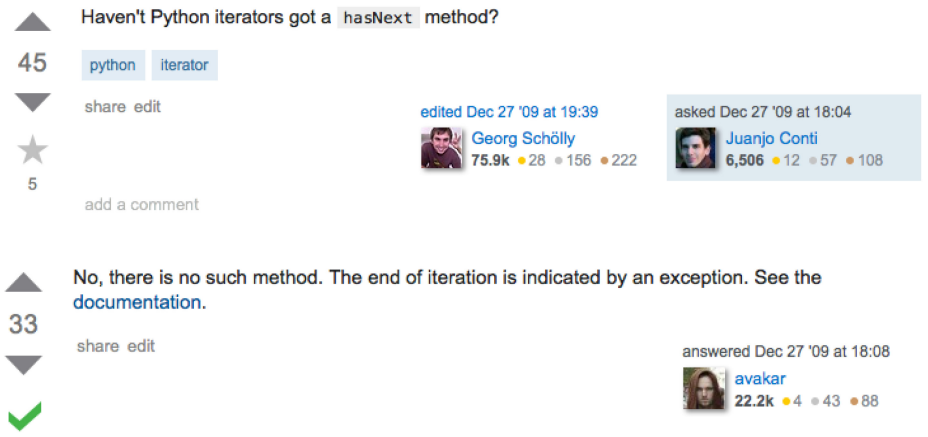


Fig. 2. A question on Python programming.

different semantic levels through the correlation between the semantics of tags and their occurrence frequencies. Through post-tag co-clustering at each tag level, we recommend different levels of tags to a user-posted question. This allows us to use some recommended tags to locate the proper KBs while using others to search/rank the matching articles thereof.
- We develop a **Knowledge Retrieval** (KR) approach, which leverages the recommended tag information to accurately retrieve the knowledge rich articles from external KBs.
- We employ a number of **Data Reduction** (DR) strategies, including dimensionality reduction for text data representation and tag selection to help the proposed approach scale to a huge volume of data.
- We conduct extensive experiments over real-world data from a Q&A site (i.e., StackOverflow) and multiple KBs to evaluate the effectiveness of the proposed framework.

The remainder of the paper is organized as follows. In Section 2, we review and discuss some representative related work. In Section 3, we outline the theoretical foundation for post-tag co-clustering. In Section 4, we describe in detail the proposed framework for KB article retrieval. In Section 5, we show our experimental results over real-world StackOverflow data. We conclude the article and point out some important future directions in Section 6.

2 RELATED WORK

We discuss some representative related research efforts, which can be classified into *query-answering systems* and *tag recommendation for Stack Overflow*.

2.1 Query-answering Systems

Knowledge retrieval for question answering is a longstanding research issue that has been intensively investigated. The majority of this body of work relies on curated or structured KBs and formal query languages to make matching between natural language questions and answers. Reference [5] uses an ontology-based KB to assist users by relating their questions to the concepts, creating context for their questions, and inferring the required information to be included in the answers. Reference [4] proposes an ontology-based approach to derive patterns to process questions, modeling them as a collection of assertions and queries related to the KB. Users will then choose query frames based on the presented patterns. The system computes F-logic queried based on the frames and processes those queries through an ontology-based access method. Reference [11]

proposes to leverage both curated KBs such as Freebase or KBs automatically extracted from unstructured text through Open Information Extraction. The questions are rewritten in terms of queries and the potential answers are generated through keyword matching and string similarity comparison along with the maximization of matching confidence. Reference [17] proposes a question-answer matching approach that combines word-level and parse tree-level similarity. The word level matching is performed through IR engines and CONTEX parser. The parse tree-level matching is done through a QA Typology, which classifies the types of questions and the use of WordNet, a concept ontology typically used as a cognitive synonyms lexicon.

Reference [24] proposes to answer questions through crawling web tables and leveraging their hidden data schema to derive the entities and their relationships. It follows two steps to generate answers for a question. The first step focusing on topic matching between questions and web tables, where deep neural networks are used to perform fine-grained matching. The second step focusing on locating the answers by checking the returning results from a search engine. Reference [1] proposes to integrate semantic parsing and question answering together instead of treating them as separate steps. It models the problem as a set of translation steps. A question is decomposed into several single-relation spans and each span is converted into formal triplets regarding the KBs. The answers are generated through the maximization of matching probability and minimization of accumulated errors. Reference [25] deals with the potential ambiguity issues caused by simple queries. It crawls search logs of a curated KB, Freebase, offline and mines query templates. It models questions as subject-relation pairs and selects relevant query templates for them to instantiate queries, which will be processed in a web search method to retrieve answers. Reference [23] crawls online unstructured KBs and stores the information in a structured database. The attributes of the entities in KBs and their links to others are kept through the database tables. Answering questions are performed through the mapping of a question to a set of queries and a confidence score reflecting the likelihood of generating the required answers. Reference [8] proposes a learn-to-rank (L2R) method to rank and recommend answers to questions in Q&A forums such as Stack Overflow. It incorporates multiple types of features such as those related to users, posts, user graphs, review styles, and readibility, and it uses those features to predict the ratings of answers using random forests.

All of the above systems focusing on leveraging the structured knowledge from KBs to answer questions. Our work, however, focusing on dealing with unstructured KBs.

2.2 Tag Recommendation for Stack Overflow

Reference [21] proposes an approach to automatically recommend tags for new Stack Overflow posts. It represents each post as a term vector based on the terms in the post. It selects 843 popular tags and solve the tag recommendation problem through multi-label classification, i.e., each tag is considered as a post label. Support vector machine (SVM) is used for the classification. Reference [19] adopts n-gram to build feature space for Stack Overflow posts. This is to address the issue of traditional term vector-based approaches where the order between terms is not considered. It also solve the tag recommendation problem based on classification but instead uses a neural network to improve the recommendation accuracy. Both of these two methods recommend tags through classification and would run into complexity issues when the number of tags is huge, which is usually the case in real world scenario. Reference [29] proposes an indexing mechanism to deal with the increasing number of posts and tags. It computes the similarity between a new post and existing ones based on a term frequency method and recommend the tags from those most similar posts. This work fails to achieve a high accuracy due to it overlooks other factors that are related to similarity measurement.

Reference [22] proposes a collaborative filtering (CF)-based approach to recommend tags based on users interests and topical folksonomy. The folksonomy captures user profiles, tags, topics, and their relationships. It applies a topic modeling technique, LDA, to learn the latent topics of posts and models users as a topic vector based on their interests. Tags are linked to topics based on their usage frequencies for the topics. Our work captures the post-tag relationships in a different way. We leverage the tag descriptions and learned the relationships through the latent topics of both posts and tags. Moreover, instead of assuming a flat structure among tags, our work builds the hierarchical tag relationships, which cannot only recommend popular tags at the high level but also less frequently used but topic specific ones at lower levels.

3 PRELIMINARIES

The proposed multi-level tag recommendation builds upon and makes unique extensions to the spectral co-clustering framework, which has been demonstrated to be effective in dealing with large-scale unstructured text corpora to simultaneously cluster documents and their associated words [9]. To facilitate the later discussions, we provide an overview on spectral co-clustering, focusing on its theoretical foundation.

Given two sets of objects: $\mathcal{T} = \{t_1, \ldots, t_N\}$ and $\mathcal{P} = \{p_1, \ldots, p_M\}$, which represent tags and posts in the context of our discussion. A bipartite graph $G = (\mathcal{T}, \mathcal{P}, W)$ can be constructed, where W_{ij} is the weight of the edge between t_i and p_j, reflecting their closeness or similarity. In a typical document-word bipartite graph, the edge weight can be naturally represented using the occurrence frequency of the word in the document. We will discuss in detail how to establish the edge weights in our tag-post bipartite graph in next section. Since there are no edges among objects of the same type, the adjacency matrix of bipartite graph is given by

$$A = \begin{bmatrix} 0 & W \\ W^T & 0 \end{bmatrix}, \tag{1}$$

where $A \in \mathbb{R}^{(M+N)\times(M+N)}$. We further define the degree of a vertex as the sum of the weights of the edges linked to it: $d(v_i) = \sum_{j=1}^{M+N} W_{ij}$. As a result, we have the degree matrix $D = \text{diag}\{d_1, \ldots, d_{M+N}\}$.

Having the bipartite graph, simultaneously clustering the two types of objects (e.g., document vs. word or tag vs. post in our case) can be achieved through graph partitioning, which is equivalent to cutting off edges from $G = (\mathcal{T}, \mathcal{P}, W)$. Assume that we aim to generate K clusters $C_1, \ldots, C_K$. Let $W(C_p, C_q) = \sum_{i\in C_p, j\in C_q} W_{ij}$ denote the cut between two clusters. As the goal of clustering is to remove the weakly connected edges while keeping the strongly connected ones, a good clustering can be achieved by minimizing the overall cut as a result of graph partitioning, given by

$$\text{Cut}(C_1, \ldots, C_K) = \frac{1}{2}\sum_{k=1}^{K} W(C_k, \bar{C_k}), \tag{2}$$

where $\bar{C_k}$ is the complementary set of C_k.

Without any additional constraint, directly minimizing Equation (2) will lead to unbalanced partitioning, which tries to assign a single vertex to each of the $K-1$ clusters and keep the remaining vertices in one cluster. Normalized cut can be applied to achieve a more balanced clustering assignment. Let $vol(C_k)$ denote the total sum of edge weights within cluster C_k. In essence, $vol(C_k)$ measures the size of C_k. Introducing $vol(C_k)$ as a penalty factor to prevent from unbalanced partitioning, the normalized cut cost function is given by

$$\text{nCut}(C_1, \ldots, C_K) = \frac{1}{2}\sum_{k=1}^{K} \frac{W(C_k, \bar{C_k})}{vol(C_k)} = \sum_{k=1}^{K} \frac{\text{Cut}(C_k, \bar{C_k})}{vol(C_k)}. \tag{3}$$

However solving Equation (3) exactly is NP-hard. To bypass the high computational cost, we instead seek for a solution that can be computed much more efficiently while providing a good approximation to the true optimal of Equation (3).

We start by constructing the Laplacian matrix of the bipartite graph:

$$L = D - A = \begin{bmatrix} D_1 & -W \\ -W^T & D_2 \end{bmatrix}, \tag{4}$$

where $D_1 = \text{diag}\{d_1, \ldots, d_M\}$ and $D_2 = \text{diag}\{d_{M+1}, \ldots, d_{M+N}\}$. The Laplacian matrix L has two important properties, which are useful for finding the approximate graph partition solution: (1) *L is semi-definite*; and (2) *The minimum eigenvalue of L is 0.* (1) can be easily derived as for any $\mathbf{f} \in \mathbb{R}^{M+N}$:

$$\mathbf{f}^T L \mathbf{f} = \frac{1}{2} \sum_{i,j=1}^{M+N} W_{ij}(f_i - f_j)^2 \geq 0 \tag{5}$$

For Equation (2), recall that $d_i = \sum_j W_{ij}$. Given a unit vector $\mathbf{e} = (1, \ldots, 1)^T$ of size $(M + N)$, it is clear that $L\mathbf{e} = 0$. Due to Equation (1), all the eigenvalues of L are nonnegative. Hence, the minimum eigenvalue of L is 0.

We are now ready to discuss the clustering process. Assume that we have K indicator vectors $\{\mathbf{h}_1, \ldots, \mathbf{h}_K\}$, which assign the $(M + N)$ objects into K clusters. In particular, we have $\mathbf{h}_k \in \mathbb{R}^{M+N}$ and let

$$h_{kn} = \begin{cases} \frac{1}{\sqrt{vol(C_k)}} & \text{if} \quad v_n \in C_k \\ 0 & \text{if} \quad v_n \notin C_k \end{cases}. \tag{6}$$

From Equation (6), we have $\mathbf{h}_i^T \mathbf{h}_j = 0, \forall i, j \in [1, .., K], i \neq j$ and $\mathbf{h}_i^T \mathbf{h}_i = \frac{1}{vol(C_i)}$. Let matrix $H = (\mathbf{h}_1, \ldots, \mathbf{h}_K)$, and we have

$$H^{-1} D H = H^T D H = I. \tag{7}$$

Given the definition of the Laplacian matrix L, we have

$$\mathbf{h}_k^T L \mathbf{h}_k = \frac{cut(C_k, \bar{C_k})}{vol(C_k)}. \tag{8}$$

Note that Equation (8) is exactly the kth component of the normalized cut cost function Equation (3). Hence, we have

$$\text{nCut}(C_1, \ldots, C_K) = \sum_{k=1}^{K} \mathbf{h}_k^T L \mathbf{h}_k = \text{Tr}(H^T L H), \tag{9}$$

where $\text{Tr}(X)$ is the trace of matrix X. Thus, we transform the clustering problem into solving the K indicator vector matrix H, which is equivalent to

$$\min_H \text{Tr}(H^T L H) \qquad \text{subject to} \quad H^T D H = I. \tag{10}$$

Solving Equation (10) is equivalent to solving the generalized eigenvalue problem

$$\begin{aligned} & L\mathbf{h} = \lambda D\mathbf{h} \\ \Rightarrow & \\ & \begin{bmatrix} D_1 & -W \\ -W^T & D_2 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \lambda \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \\ \Rightarrow & \\ & \begin{cases} D_1^{\frac{1}{2}}\mathbf{x} - D_1^{-\frac{1}{2}} W\mathbf{y} = \lambda D_1^{\frac{1}{2}}\mathbf{x} \\ -D_2^{-\frac{1}{2}} W^T\mathbf{x} + D_2^{\frac{1}{2}}\mathbf{y} = \lambda D_2^{\frac{1}{2}}\mathbf{y} \end{cases}, \end{aligned} \tag{11}$$

where $\mathbf{h} = (\mathbf{x}, \mathbf{y})^T$. Let $\mathbf{u} = D_1^{\frac{1}{2}}\mathbf{x}$ and $\mathbf{v} = D_2^{\frac{1}{2}}\mathbf{y}$. The generalized eigenvalue problem can be more efficiently tackled by solving a singular value decomposition problem with a much smaller size:

$$\begin{cases} D_1^{-\frac{1}{2}} W D_2^{-\frac{1}{2}}\mathbf{v} = \sigma\mathbf{u} \\ D_2^{-\frac{1}{2}} W^T D_2^{-\frac{1}{2}}\mathbf{u} = \sigma\mathbf{v} \end{cases}, \tag{12}$$

where $\sigma = (1 - \lambda)$. We can use the first K left and right singular vectors of matrix $D_1^{-\frac{1}{2}} W D_2^{-\frac{1}{2}}$ to approximate the discrete indicator vectors. The final clusters can be achieved by applying simple clustering algorithms (e.g., K-means) to these approximated indicator vectors.

4 THE KB RETRIEVAL FRAMEWORK

We present the proposed knowledge retrieval framework in this section. We start by presenting the overall system architecture, which gives a high-level overview of the framework. We then elaborate on the two key components of the framework: multi-level tag recommendation (MLTR) and knowledge retrieval (KR).

Figure 3 shows the overall system architecture. When a user posts a question, the post will be first processed by the MLTR component. During the training process, MLTR separates the tags associated with the historical user posts into multiple levels (e.g., hot and common tags) and constructs post-tag co-clusters at each level. Then, a supervised classifier is trained based on the clusters at each level. When a new question is posted by a user, it is first classified by each of these classifiers into one of the clusters at each level. Then, a set of similar historical posts are identified within the corresponding clusters and their user-assigned tags are used to recommend to the new post. The KR component uses the recommended hot tags to select the appropriate KBs from multiple candidates and other recommended tags to match and rank the articles in the selected KBs.

4.1 Multi-level Tag Recommendation

Tags in the Q&A sites, especially those related to software development, usually come with relatively rich descriptive information. For example, most tags in StackOverflow are described by two individual child posts: excerpt and wiki. The excerpt part provides some general introduction of the tag along with some expected questions. The wiki part usually includes more detailed information, such as commonly used libraries and tools, historical versions, code samples, question samples, most frequent questions, similar tags, and references like external links and text books.

4.1.1 Computing the Weight Matrix W. As described in Section 3, to simultaneously cluster both posts and tags, a key step is to compute the edge weight for each post-tag pair. Given the rich descriptions of tags, we can use the classical term-based vector space model to represent both

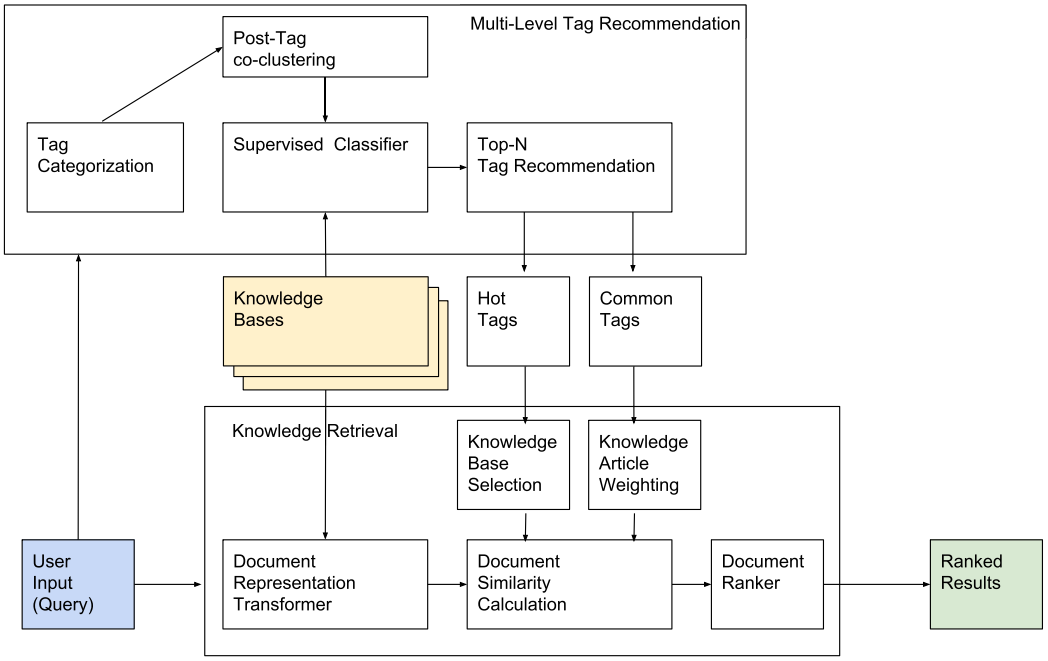


Fig. 3. System architecture.

posts and tags as term vectors. In particular, assume that we have a global vocabulary V, which is composed of all the distinct terms from the posts and tag descriptions. Hence, the i-th post and the j-th tag can be denoted as $\mathbf{p}_i \in \mathbb{R}^{|V|}$ and $\mathbf{t}_j \in \mathbb{R}^{|V|}$, respectively. Furthermore, p_{ik} and t_{jk} are set to the tf-idf of the kth term in the post and the tag, respectively. This will allow us to compute $W_{ij} = \mathbf{p}_i^T \mathbf{t}_j$ (or use cosine similarity by first normalizing $\mathbf{p}_i$ and $\mathbf{t}_j$). Therefore, the entire weight matrix can be computed as $W = P^T T$, where $P = (\mathbf{p}_1, \ldots, \mathbf{p}_M)$ and $T = (\mathbf{t}_1, \ldots, \mathbf{t}_N)$.

A major issue with the term-based vector space representation is the high computational cost especially when scaling to a large number of posts and tags. It requires to store a large term vocabulary V and the complexity of computing W is $O(MN|V|)$, which increases linearly with the size of V. As W_{ij} is mainly used to capture the semantic similarity between the ith post and the j-th tag, we may exploit some dimensionality reduction approach to first project the large term vectors onto a low-dimensional space while keeping the important semantics. This will allow us to compute the post-tag similarity in the low-dimensional semantic space. We propose to employ Latent Dirichlet Allocation (LDA) [2] to achieve this purpose. A key advantage of LDA over other dimensionality reduction models is that it produces human interpretable latent semantics, referred to as topics, where each topic is a distribution over terms in the term vocabulary. Through LDA, a post $\mathbf{p}_i$ (or tag $\mathbf{t}_j$) can be represented by a vector in a Z-dimensional topic simplex, where Z is the number of topics. In particular, $\mathbf{p}_i$ (or $\mathbf{t}_j$) $\in \mathbb{R}^Z$, where p_{ik} (or t_{jk}) denotes the proportion of the kth topic in the ith post (or jth tag). In this way, the cost of computing W is reduced to $O(MNZ)$, which is a significant improvement, since we have $Z \ll |V|$.

Through the vector space or topic models, we can establish the post-tag similarity and hence construct the bipartite graph for post-tag co-clustering. Meanwhile, we also notice that the historical posts in many Q&A sites (e.g., StackOverflow) already come with user assigned tags. As these tags were carefully chosen by human users, it is beneficial to include this useful prior knowledge

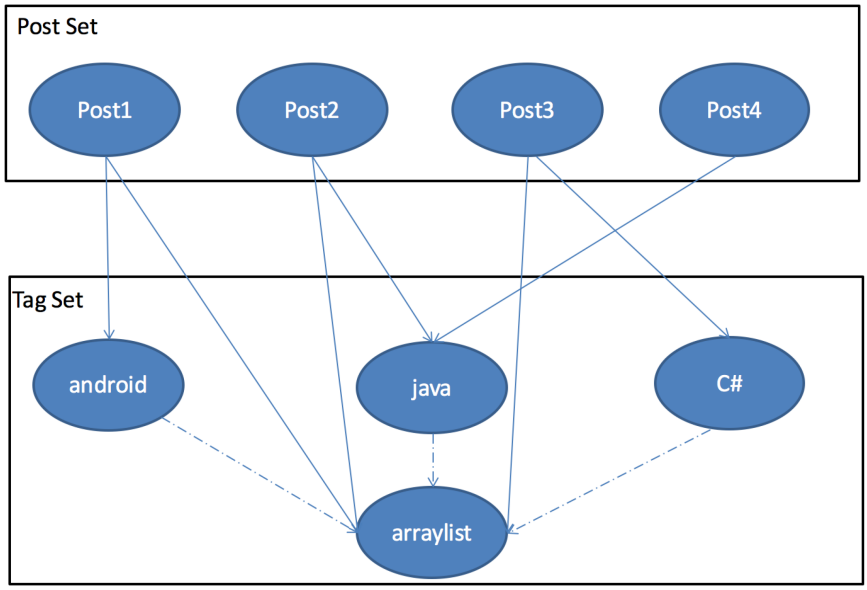


Fig. 4. Hierarchical structure of tags.

to construct the post-tag bipartite graph. In particular, if the jth tag is assigned to the ith post by a user, it is reasonable to believe that they are semantically related, which should be captured by the edge weight W_{ij}. We propose the following strategy to incorporate this prior knowledge:

$$W_{ij} = W_{ij} \times \log \text{itf}_j \quad \text{if tag}_j \text{ is assigned to post}_i, \tag{13}$$

where itf_j is the inverse tag frequency of tag_j,

$$\text{itf}_j = \frac{\text{total number of posts}}{\text{number of posts assigned with tag}_j}, \tag{14}$$

which assigns higher weights to tags being rare across all posts.

4.1.2 Multi-level Post-Tag Co-clustering. The primary purpose of post-tag co-clustering is to form cohesive groups of posts and tags. For a new post, we could find the most relevant post-tag cluster and use the tags in that group for recommendation purpose. Since we do not assume any structural constraints on posts aiming to leverage free-form text data, it becomes significantly more challenging to create high-quality post-tag clusters for accurate tag recommendation.

One key challenge is the large and highly complex tag space. For example, there are over 40k tags in the StackOverflow site. Besides those identifying high-level categories (e.g., Java or Python), many tags are also used to help user specify the technical details about the questions (e.g., I/O and arraylist). As a result, different tags may convey different levels of semantics and hence form a hierarchical structure instead. As the vector space or the LDA-based topic models assume a flat structure on the terms or the latent topics, they could lead to inappropriate grouping of posts and tags during the co-clustering process.

Figure 4 illustrates this issue using StackOverflow tags. The arraylist tag represents a detailed technical concept, which may appear in many programming languages or platforms, such as android, java, and C#. This implies that it is on a lower semantic level than other tags (as indicated by the dashed lines in the figure). A solid line represents that a tag has been assigned to a corresponding post by a user (e.g., Post1 has android as a tag). Ignoring such a hidden hierarchical

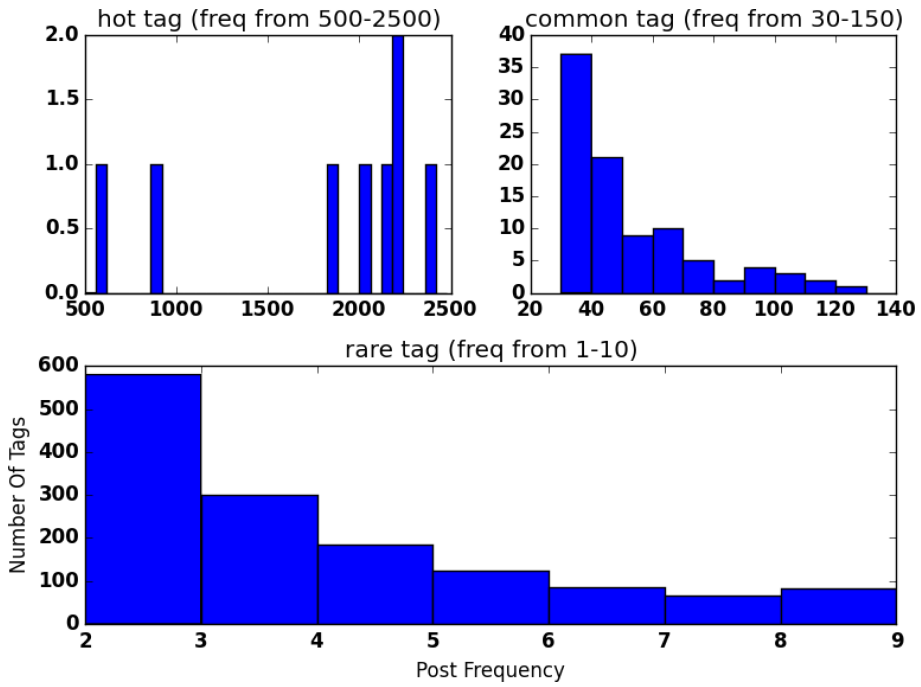


Fig. 5. An example of the tag frequency levels.

structure, arraylist would be assigned into the same cluster with one of the higher-level tags that it frequently co-occurs with, such as java as they co-occur in Post2 and Post4. This will cause arraylist to be separated from android and C#, which are the languages using arraylist as well. Hence, for posts fall into these two languages, there is no chance to recommend arraylist as a tag even though it may be a highly relevant one.

We address the above issue through multi-level post-tag co-clustering, where tags are organized into multiple semantic levels (see below for details). The idea is that we build a post-tag co-clustering model at each level. For each model, we include all the posts but only the tags in that particular level. This will allow us to group a post with tags at different semantic levels. To further illustrate, for the simple example in Figure 4, we can separate the tags into two levels: level 1 (android, java, and C#) and level 2 (arraylist). In level 1 co-clustering, we create three clusters: (Post1, android), (Post2, Post4, java), and (Post3, C#). In level 2 co-clustering, we create two clusters: (Post1, Post2, Post3, arraylist) and (Post4). In this way, for a post that asks a question about arraylist in C#, multi-level post-tag co-clustering will allow us to recommend both C# and arraylist as relevant tags.

Now the only remaining issue is how to separate tags into different levels. Apparently, this can be manually done by a group of domain experts. Instead, we propose a simple yet effective way to automate this process based on an important observation. It turns out that the semantic levels of tags have a strong correlation with their usage frequencies. Figure 5 shows the frequencies of StackOverflow tags. Based on the frequencies of these tags, they fall naturally into three categories: *hot*, *common*, and *rare*. As can be seen, there are some obvious gaps in terms of their frequencies between tags in these categories.

We have manually conducted a thorough investigation on the key properties of the tags in each of these categories and summarize our results in Table 1. By analyzing these results, we reach two

Table 1. Tag Categories and Key Properties

Category (Examples)	**Key Properties**
Hot: Javascript, java, C#, php, android, python, html, C++, ios	1. knowledge domain, 2. integrated develop environment, 3. popular programming language, 4. usually independent from other hot tags
Common: jquery, cmd, events, networking, serialization, recursion, thread-safe, multithreading	1. common operation, 2. popular technique, 3. further description under hot tags, 4. usually shared by multiple hot tags
Rare: metacity, entourage, resig, ubiquity, smarthost, array-initialize, netmon, ora-01652	1. out-of-date technique, 2. niche software/tool, 3. uncommon terminology, 4. content of the tag usually poorly maintained

important conclusions. First, the hot and common categories provide a good approximation of two semantic levels of tags, where the higher level corresponds to domain, language, or platform and the lower level corresponds to the key techniques within the domain, language, or platform. Given that many rare tags are outdated or less commonly used, we could ignore them during the co-clustering process. This will significantly reduce both the computational and spatial cost given the large number of rare tags. Despite not being considered by multi-level post-tag co-clustering, highly relevant rare tags may still be recommended through our tag recommendation algorithm, which will be described next.

4.1.3 Multi-level Tag Recommendation. Given a new post (with no tags), a straightforward way to recommend tags is to first locate clusters at each semantic level and then recommend all the tags within the identified clusters. There are two issues with such an approach. First, if the sizes of the clusters are large, a large number of tags will be recommended, which may include some not very relevant ones. Second, since we only build co-clustering models for hot and common tags, there is no way to recommend highly relevant rare tags.

To address the issue, we propose to recommend tags through two steps. First, to leverage the grouping information from post-tag co-clustering, we train a supervised classifier (SVM is used in our experiments) for each level using the clusters as class labels. Second, we classify the new post into one of the clusters at each level. Within the identified cluster, we choose the top-n most similar posts and use the top-k most frequent tags of these posts for recommendation purpose. This two-step approach is able to leverage both global structure of the data (through post-tag co-clustering) and local neighborhood information (by searching the most similar historical posts within the cluster) to achieve high recommendation accuracy. Furthermore, it may recommend relevant rare tags when these tags are frequently assigned to the chosen similar posts.

Regarding the time complexity, at the kth level of tags, we need to first conduct the post-tag co-clustering, which applies singular value decomposition to a M_k by N_k matrix, where M_k and N_k are number of posts and tags at the kth level, respectively. This step has the complexity of $O(M_k^2 N_k)$. Next, we use all posts at the current level and their belonged clusters as labels to fit a multi-class SVM classifier. The multi-class SVM classifier adopts one-vs.-rest classification strategy thus consists of L_k binary SVMs where L_k is the number of clusters at kth level. Each binary SVM is optimized using sequential minimal optimization with the complexity of $O(M_k^2)$. The total training complexity at the kth level of tags is therefore $O(L_k {M_k}^2)$. However, the training of the tag recommender can be performed offline. For real-time tag recommendation, the key operation is to invoke the multi-class SVM for label prediction, which can be conducted very efficiently with a performance linear to the number of support vectors.

4.2 Knowledge Retrieval

The knowledge retrieval (KR) component of the framework performs two major tasks: selecting appropriate KBs and then ranking matching documents within these KBs. In the following subsections, we describe each of these tasks.

4.2.1 Knowledge Base Selection. Selecting appropriate KBs is critical to retrieve relevant KB articles to answer user posted questions. This can be significantly facilitated using the tags recommended by the MLTR component as described in Section 4.1. In particular, as hot tags usually correspond to knowledge domains, programming languages, or developer platforms, they can be naturally used to match the main themes of the KBs. This observation allows us to assign a hot tag to each of the available KBs beforehand (e.g., assign javascript to online JavaScript Reference and php to online PHP Documentation). Then, we can use the recommended hot tags to choose the appropriate KBs. We refer to all of these selected hot tags as *knowledge base or KB tags*.

We need to consider three possible scenarios: (1) One and only one KB tag is recommended. In this case, we directly find the matching KB. (2) More than one KB tags are recommended. Although this is rare as evidenced by our experimental results, it may still occur in practice. For example, a user may explain what s/he can do in one language (e.g., Java) and ask how the same task can be achieved using another language (e.g., python). In this case, it makes sense to retrieval relevant articles from both KBs, where the articles from the first KB can serve as a reference to facilitate the user to understand the articles from the second KB. (3) No KB tag is recommended. In this case, we can always go back to the MLTR component to recommend more tags. However, as the recommendation quality becomes lower, it may imply either the user is asking for a subject that is not covered by any available KBs or the posted question is not properly formulated. In this case, some feedback can be sent to the users to inform them the potential issues with their posts.

4.2.2 Retrieving and Ranking KB Articles. Because we wish to identify articles from a set of chosen knowledge bases, we tackled this problem as an information retrieval task. By ranking these articles by a similarity metric, we would be able to follow in this direction. While our MLTR were able to leverage both high and low-level tag recommendations, for the purpose of ranking, we only used the high-level tag recommendation, because we could not assure an accurate relationship between a choosen KB set and a low-level tag for individual KB articles. For the purpose of tag prediction, we processed the MLTR component for all KB articles. However time-consuming this procedure might be, this tag recommendation procedure for KB articles can be conducted offline beforehand, prior to the query processing, so that all articles have already been assigned tags before users query online.

Since tags are predicted for both the newly posted question and each KB article, a straightforward way of retrieving relevant KB articles is to rank the articles using the Jaccard Coefficient computed from their corresponding sets of tags, where $\text{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$ for sets A and B.

However, there is a potential issue with this strategy. Note that tag prediction is based on historical tags assigned by Q&A community users. A small number of tags may be able to capture the underlying semantics of a short question. However, they may not be sufficient for capturing all the important details of KB articles with much longer content. Thus, simply performing tag matching may miss important KB articles that are relevant to answering the question.

By taking advantage of the tag suggestions provided by our MLTR component, we effectively augmented the term representation of our knowledge base articles. The size of the article varies, but this is particularly beneficial when dealing with short articles. Table 2 presents some example KB articles and the recommended tags. The examples clearly demonstrate the semantic relevance between the articles and the recommended tags.

Table 2. KB Articles and Recommended Tags Examples

Knowledge Base Articles	Recommended Tags
Resource Types This extension defines a XML-RPC server resource returned by xmlrpc_server_create (...)	xml, android-layout url, spring, xml, mysql web-applications, eclipse, file, regex
Once-only subpatterns With both maximizing and minimizing repetition failure of what follows normally causes the repeated item (...)	regex, replace, split string, python-3.x, c++, design-patterns, performance, if-statement
Recursive patterns Consider the problem of matching a string in parentheses allowing for unlimited nested parentheses (...)	design-patterns, replace, loops split, regex, string, loops recursion, algorithm, c++

We propose to adopt a weighted query expansion strategy. We append the recommended tags to the text of both the posted question and the KB article. Inspired by the tfidf heuristic, we chose to increase the weight of a recommended tag by multiplying its term frequency in the question (and the KB article) by its inverse tag frequency (or itf) in the corresponding KB, which is given by

$$\text{itf}_j = \frac{\text{total number of articles in the KB}}{\text{number of articles assigned with tag}_j}. \tag{15}$$

This strategy makes questions and KB articles that share the same tags more similar and hence improve the ranking of these articles.

We adopt the term-based vectors to represent the question and all the KB articles, where the standard tf-idf weighting can be used to quantify the importance of each term in the question (and KB articles). Articles in the selected KBs can be ranked based on their cosine similarities with the question and the top ranked ones will be returned to the user. There are two major reasons for choosing a term vector-based presentation instead of a topic-based representation for KB article retrieval. First, since the tags are predicted through the topic-based representation, they are expected to already capture the high-level semantics conveyed through the topic-based presentation. Leveraging the term vector-based representation will allow us to also incorporate some important low-level details conveyed through individual terms. Second, through KB selection, we reduce the KB article search scope to one or a small number of KBs. Since we only need to compare a relatively small number of KB articles, search efficiency can still be guaranteed with term-based presentation.

Regarding the time complexity of the Knowledge Retrieval component, we need to build the tf-idf representation and an inverted index of a KB. Assuming we have D articles with a total W terms, the complexity is $O(W)$. This step, however, can be performed offline. Through an inverted index, the matching between a query and potential KB articles can be performed very efficiently, which is upper bounded by $O(D)$ (when a question consisting of keywords that match all the KB articles, which is unlikely to occur).

5 EXPERIMENTS

We have conducted extensive experiments over real-world data collected from StackOverflow. The reason of choosing StackOverflow is due to its great popularity, large user base, huge amounts of historical posts from users, and rich set of well-maintained tags. Furthermore, we choose six well-known online API repositories as our potential KBs. The proposed approach can be conveniently generalized to many other Q&A sites with a similar nature. We start by describing the experimental data in detail. We then define important evaluation metrics and present our experimental result and comparisons with the baseline and other alternative approaches.

Table 3. Summary Statistics of StackOverflow Data

Statistic	Original	Experiment Set
Posts	2,430,480	64,793
Questions	534,611	12,000
Answers	1,895,869	52,793
Tags	15,379	3,247

Table 4. Summary Statistics of Processed KBs

Knowledge Base Title	Article Count
Python KB (PSL + PLR combined)	5,469
Java KB (JPAS + TJLS combined)	1,994
PHP Documentation	12,843
Javascript Reference	790
Android Reference	5,469
Django Documentation	324

5.1 Description of the Dataset

The StackOverflow team periodically publishes updated data dumps with its content [10]. We adopt the version published on August 18, 2015, which was the most up-to-date one when we conducted this research. The StackOverflow questions and answers are represented in the dataset individually as posts. The dataset has a score metric for each post that we used for data cleansing. The score is a user voting measure and is simply the number of positive votes minus the number of negative votes. To ensure the quality of our dataset, we extracted answers with a minimum score of 1. We took questions with a minimum score of 1 and with a minimum of four answers among the already-filtered list of answers. We then randomly sampled 2,000 questions for each knowledge base with matching tags, totalling 12,000 questions along with over 50,000 linked answers. Table 3 shows the summary statistics of the dataset.

As discussed in Section 1, we choose online programming documentation libraries as external KBs. In particular, we picked the following eight data sources: Python Standard Library [14], Python Language Reference [13], Java Platform SE 8 API Specification [7], Java Language Specification, Java SE 8 Edition [6], PHP Documentation [16], JavaScript Reference [15], Android Reference [18], and Django Documentation [12]. We combine two or more sources into one if they cover the same subject (e.g., Python Standard Library and Python Language Reference), resulting in 6KBs. All references were downloaded and pre-processed. When no plain text was available the reference was parsed to extract the plain text. When only HTML documentation was available, we removed its structural tags, scripts and styling. Each resulting KB was stored as a single text file, with one article per line. Table 4 presents summary statistics of the processed KBs.

5.2 Tag Recommendation Evaluation

5.2.1 Metrics to Evaluate MLTR. To evaluate the quality of the recommended tags, we vary the number of recommendations and report the recall@K to evaluate how well the predicted tags match the user assigned ones for a given set of testing posts. For each post in the testing set, we first remove the original user-assigned tags. Once the tags are predicted, we essentially use the user-assigned tags as our ground truth to compute the recall. Since the tags are assigned by a diverse set of users, we may not expect all these tags are truly relevant and some of them may also be

redundant. More importantly, the user assigned tags may not fully cover the underlying semantics of the posts. Therefore, precision is not a good choice for evaluation of our recommendation result. Later in our discussion, we will show that MLTR is able to recommend some novel tags that are truly relevant to the posts but are not assigned by users.

The noisy nature of user assigned tags further motivates us to define a more robust evaluation metric, referred to as *Coverage*, which allows the predicted tags to only partially match the user assigned tags. Consider a set of testing posts $\mathcal{P}$. For a given post $p_i \in \mathcal{P}$, let $\mathcal{R}_i$ denote the recommended tags from different levels and O_i denote the original tags assigned by users. We define an indicator variable cover_i to show whether $\mathcal{R}_i$ overlaps O_i:

$$\text{cover}_i = \begin{cases} 1 & \mathcal{R}_i \cap O_i \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

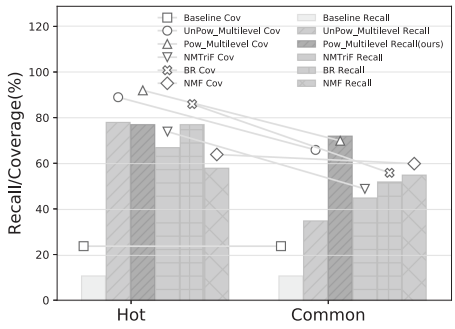
The Coverage is then defined as

$$\text{Coverage} = \frac{\sum_{i=1}^{N} \text{cover}_i}{N}. \tag{17}$$

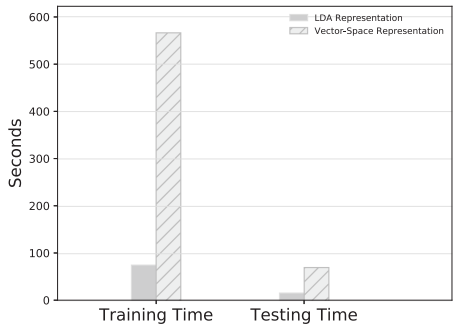
5.2.2 Results and Comparisons. We adopt tenfold cross-validation to evaluate the quality of multi-level tag recommendation. To show the effectiveness of the proposed approach, we apply post-tag co-clustering to tags with a flat structure (i.e., no separation into different levels) and use it as the baseline for comparison. To have a more detailed view on the result, we evaluate the recommendation accuracy at each of the tag levels: hot and common. The Coverage (or Recall) is computed by comparing the user assigned tags with the predicted ones at the same level based on Equation (17) (or the Recall function). We also investigate the effectiveness of using inverse tag frequency to adjust the graph edge weights according to user assigned tags in the training set. We set the number of clusters (at each level) as 12, number of topics of LDA as 12, and number of recommended tags (at each level) as 5.

Figure 6(a) summarizes the results. First, multi-level recommendation significantly outperforms the baseline approach at both levels. Second, adjusting the edge weights (denoted as Pow_Multilevel, the darkest bar in the figure) has a large positive impact on common tag recommendation. Different from hot tags, common tags usually have shorter and sometime insufficient descriptions. By leveraging users' past usage of these tags, our approach essentially combines both content-based and collaborative filtering recommendation to achieve high accuracy. Third, the proposed approach achieves high recall and coverage, particularly for common tags. The relatively short descriptions and lack of intensive usage by users make common tag recommendation more challenging but extremely important, because choosing the right common tags can make a post easily noticed by experts from its corresponding domain (usually indicated by the hot tag). Furthermore, relevant common tags may directly lead a user to the solution. For example, common tags **jquery** and **for-loop** recommended to the post *"convert numerical value to a string filled with a character in JavaScript"* indicate that the user should find the solution using for-loop with jquery functions. Finally, the effectiveness of multi-level co-clustering can also be seen from the shape of the clusters. Our results indicate that the flat co-clustering usually leads to very unbalanced clusters, some of which have one or two tags and even zero post. This is due to the hidden hierarchical structure of tags as illustrated in Figure 4. Multi-level co-clustering effectively addresses this issue and achieves perfectly balanced clusters (the actual cluster distributions are omitted due to the lack of space).

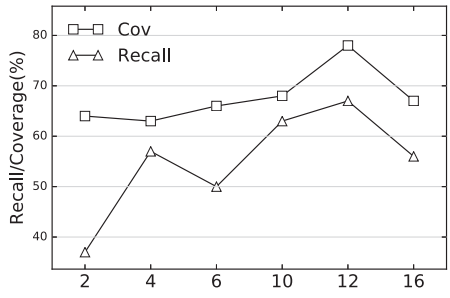
We compare the performance of the proposed recommendation method with three competitive tag recommendation methods in Figure 6(a): binary relevant machines (BR), non-negative matrix factorization (NMF), and non-negative matrix tri-factorization (NMTriF) [3]. The BR method trains



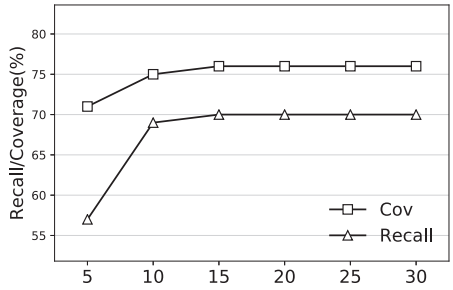
(a) Performance comparison with the baseline and using edge weight adjustment



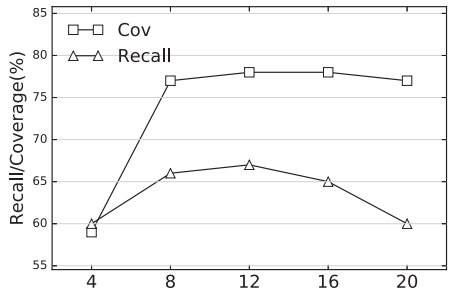
(b) Execution time for topic representation and vector space representation



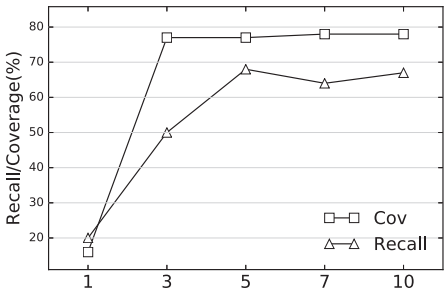
(c) Recall/Coverage Vs. Number of clusters



(d) Recall/Coverage Vs.Number of voting posts



(e) Recall/Coverage Vs. Number of topics



(f) Recall/Coverage Vs. Number of recommended tags

Fig. 6. Multi-level tag recommendation.

a binary classifier for each tag and predicts tags independently. BR method may suffer from a high computational cost when the number of tags is large. The NMF method factorizes the post-tag matrix into the product of two non-negative matrices, W and H. Matrix $W$is the k-components representation of tags/posts where the components are jointly extracted from tags and posts. Then we run K-means on W to cluster the posts and tags. The cluster assignment is used to train a supervised model to predict the cluster of a test post tags for that post using the same voting mechanism like the proposed method. In our experiment, k is set to 12, which is the optimal number of LDA topics for the proposed model. The NMTriF method factories the tag-post matrix into the

product of three non-negative matrices, G, S, and F. Matrix F is the one-out-of-k cluster assignment of posts and is used for tag recommendation like the proposed method.

All methods have reasonable performance for hot tag recommendation but the proposed method outperforms others for recommending common tags. The advantage of the proposed method on common tags shows that edge weights adjustment provides useful information on tag-post relationships that are beneficial to multi-level co-clustering.

In Figure 6(b), we compare the execution time of using topic represented data with vector space represented data. We show that the using LDA for dimensionality reduction not only improves the recommendation performance but also significantly reduces the time complexity.

We also conduct a series of experiments to investigate the impact of the key parameters of the model, which include the number of clusters (the same number is used for each tag level to simplify parameter tuning), number of topics for the LDA model to derive the topic-based representation of posts, the number of recommended tags (the same number is used for each level), and the top-n posts used for recommending tags in the designated cluster. We vary one parameter while keeping the others fixed to locate its optimal value. Figures 6(c), 6(e), 6(f), and 6(d) also show that Coverage and Recall are relatively stable over a range of these parameters. For the retrieval component task, we fixed both the number of topics and the number of cluster to 12.

Another important observation is that some recommended tags that do not match the user assigned ones might also be relevant to the posted question. For example, a user asked *how to recognize a piece of phone number through long strings* and a hot tag **python** is assigned to the post. Our approach not only recommends **python** as a hot tag but also **if-statement** and **regex** as common tags. If precision@K were used for evaluation, then these two tags are both false positives, which will decrease the precision. However, they provide the right combination with the recommended hot tag to best characterize the user posted question. In another case, a user asked about *how to submit a form without a submit button* but only assigned **php** as the tag. Our approach is able to recommend **jquery** and **ajax** (besides **php**), which are two popular techniques that address the posted question. These cases suggest that the proposed approach can recommend highly relevant and oftentimes novel tags to help user better understand the questions and locate relevant answers to address them.

5.3 Knowledge Retrieval Evaluation

5.3.1 Metrics to Evaluate KR. A key challenge to evaluate the overall performance of KB article retrieval is the lack of ground truth. Since we aim to retrieve articles from external KBs to answer questions, the user provided answers in the StackOverflow data are not directly applicable for evaluation purpose as in multi-level tag evaluation. To address this challenge, we construct two evaluation sets, *Human-judged* and *Automated*, to assess the accuracy of KB article retrieval.

The Human-judged evaluation set is manually created, which consists of 20 questions relevant for each KB, totalling 120 questions. For each question, we manually go through the KB articles and locate a set of most relevant articles, which will be used to compare with the system retrieved ones. We also develop a heuristic to create a larger automated evaluation set and then evaluate our overall system performance with a more representative testing set. Specifically, for each question in our dataset, we concatenate all the user provided answers. This concatenated answer is then compared with the articles in the corresponding KB and the top-10 most similar (e.g., by computing their cosine similarity) ones are first selected as a candidate set $\mathcal{S}$. Finally, an article is included in the Automated evaluation set only if its similarity with the concatenated user provided answers is no less than $[mean(\mathcal{S}) + sd(\mathcal{S})]$, where sd denote standard deviation.

Having these two evaluation sets, we can use the following metrics: Precision @K, Recall@K, Mean Reciprocal Ranking (MRR), and Normalized Discounted Cumulative Gain (NDCG) with

Table 5. Tenfold Cross-validation Accuracy Estimates

	Classifier							
Tags	SVC	LDA	QDA	LR	KNN	AB	GB	RF
java	50%	65%	38%	60%	60%	63%	53%	50%
javascript	50%	60%	38%	70%	48%	60%	50%	50%
php	50%	68%	55%	65%	63%	70%	50%	53%
python	50%	65%	58%	73%	58%	80%	50%	53%
android	50%	65%	63%	75%	78%	70%	50%	63%

binary (0 or 1) relevance. NDCG is particularly interesting when the order of the retrieved results is important. As our system returns a ranked list of articles, it is desired that the most highly ranked results are likely the most important ones. This behavior is most accurately quantified using NDCG, because it introduces increasing discounts or penalties the higher the rank of a result. We use the typical discount of $\frac{1}{log_2(rank)}$. Let $r_1, \ldots, r_n$ be the scores of a list of ranked articles for a given query, DCG is given by $r_1 + r_2/log_2(2) + r_3/log_2(3) + \cdots + r_n/log_2(n)$. To normalize the obtained score to achieve the NDCG@K, we divide the calculated DCG by the ideal DCG at rank K:

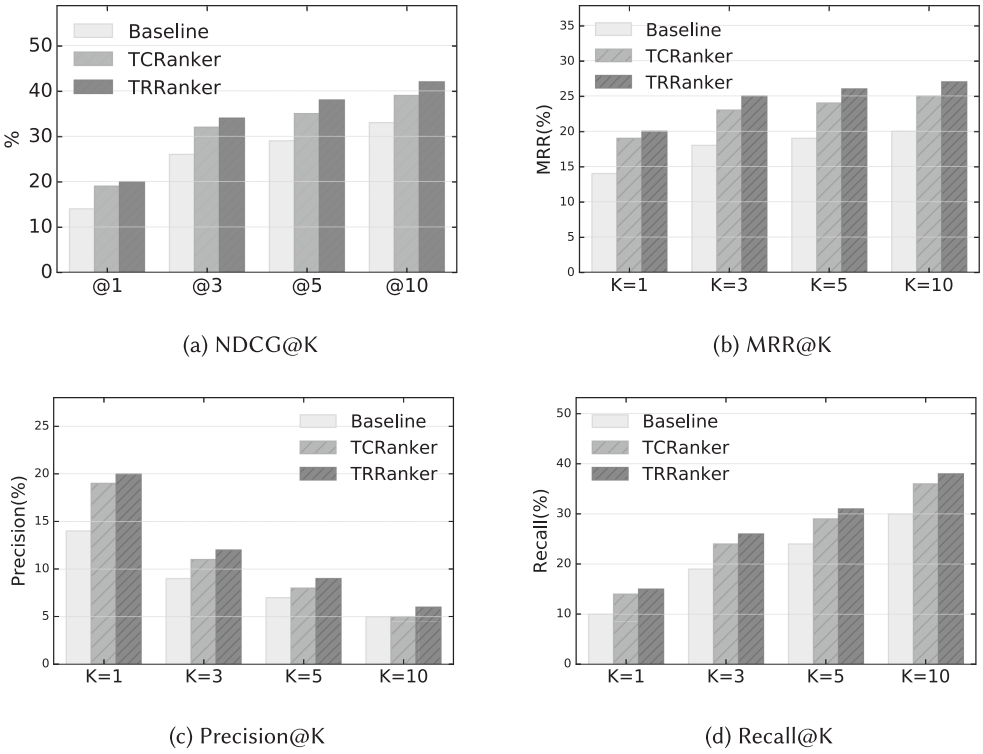
$$NDCG@K = \frac{DCG@K(\text{calculated})}{DCG@K(\text{ideal})}, \tag{18}$$

where the ideal ranking information can be derived from the constructed Human-judged or Automated evaluation sets.

5.3.2 Results and Comparisons. To justify the effectiveness of the proposed approach, we also implement two other KB article retrieval approaches. The first one, which is used as the baseline, builds a term-based vector space model over the articles of all KBs and construct an inverted index to facilitate the search of these articles. Relevant articles are identified and ranked based on their cosine similarity with the questions in the testing set. The comparison with this baseline approach will help show the effectiveness of using recommended tags for KB article retrieval. We also build a supervised classifier using the historical posts and their user assigned tags to directly predict the hot tags of a test question and use the predicted tags for article retrieval. For this classical machine-learned tag recommender, we choose an Ada Boost-based model, the most performing classifier using 10-fold cross validation. The following models were tested: Supporting Vector Machine, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Logistic Regression, K Nearest Neighbor, Ada Boosting, Gradient Boosting, and Random Forest. Table 5 summarizes each model accuracy. We refer to this classical machine learning approach as TCRanker and it was aimed at checking the effectiveness of MLTR in the context of KB article retrieval. Finally, we refer to our proposed approach as TRRanker.

For the Automated evaluation set, we follow the same 10-fold CV strategy as used in evaluating MLTR to assess the performance of KB article retrieval. For the Human-judged set, we use the entire 12,000 posts for training the system as they are distinct from the questions in the Human-judged set. Figures 7 and 8 summarize all the major results. As can be seen, for all metrics, TRRanker is able to achieve consistent improvements over the baseline and TCRanker. Both tag-based approaches outperform the baseline, which justify the effectiveness of using tags for KB article retrieval.

It is also interesting to note that the performance advantage of TRRanker is more obvious for the Human-judged set than the automatically generated one. Recall that the Automated evaluation set is generated from historical user provided answers and through some heuristics. Therefore,



(a) NDCG@K (b) MRR@K

(c) Precision@K (d) Recall@K

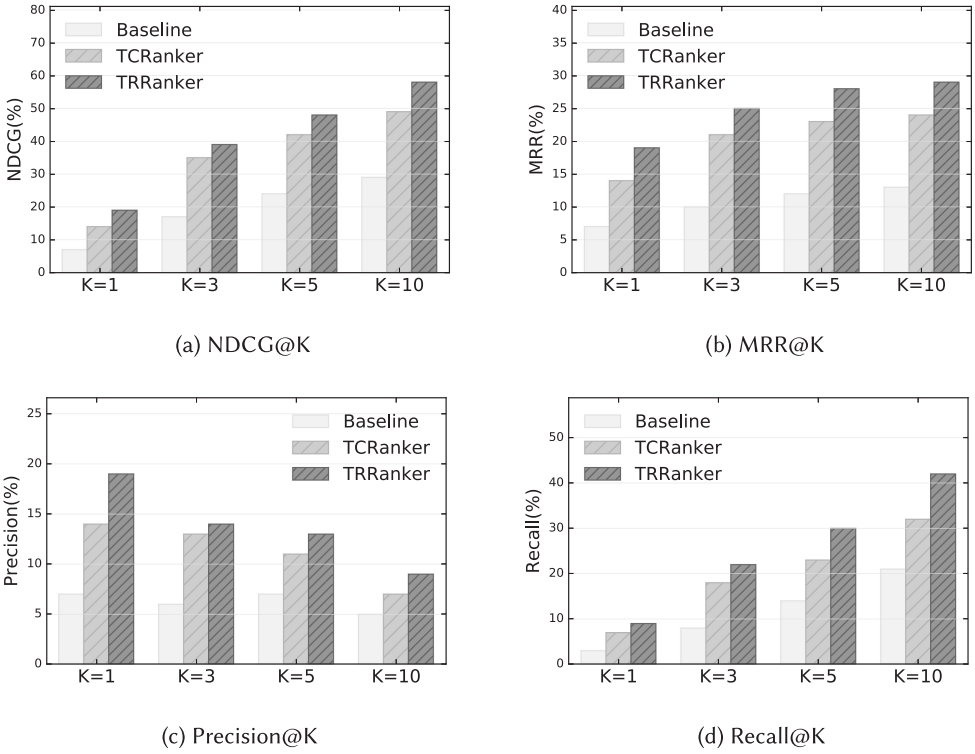
Fig. 7. Results for the automated evaluation set.

the result from the Human-judged set is expected to be more accurate. We also noticed that when applying the same heuristic to the questions in the Human-judged set, we extract a set of KB articles, which only partially match the manually selected ones. However, the good performance of our approach shows that it is able to retrieve KB articles that cover both the knowledge from other users as well as the external KBs.

Finally, we also experiment by representing both the test questions and the KB articles using LDA-based topics and fine tune the number of topics. It turns out that the term vector-based presentation consistently outperforms the topic-based one for KB article retrieval. This result justifies our analysis that the recommended tags may already carry the semantics captured by the topics (as multi-level co-clustering is achieved through the topic-based representation). Using the term-based presentation complements tags by capturing the important low-level details through term vectors.

6 CONCLUSIONS AND FUTURE WORK

We present an automatic question-answering framework by leveraging a novel multi-level tag recommendation model to retrieve relevant articles from remote knowledge bases. To best harness the rich tags assigned by Q&A community users, we categorize a large collection of tags into multiple semantic levels and automate this process by following the correlation between tag semantics and their usage frequencies. We build a post-tag co-cluster and a two-step supervised tag recommender at each tag level to handle the hidden hierarchical dependencies among tags. By leveraging the accurately recommended tags, we can locate the proper KBs and discover/rank their articles most relevant to user posted questions. We have extensively evaluated the proposed



(a) NDCG@K (b) MRR@K

(c) Precision@K (d) Recall@K

Fig. 8. Results for the human-judged evaluation set.

framework using the large-scale StackOverflow data. Results show that the performance of our framework is superior to both the baseline and other alternative approaches.

In this article, we primarily focus on software development related Q&A sites due to their great popularity and large user bases. An interesting future direction is to apply the proposed framework to other important domains, such as health related ones, or even the ones with open domains. One challenge is the identification of potential KBs and how to better scale the system when a much larger number of KBs are involved.

ACKNOWLEDGMENT

The views and conclusions contained in this article are those of the authors and should not be interpreted as representing any funding agency.

REFERENCES

[1] Junwei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao. 2014. Knowledge-based question answering as machine translation. *Cell* 2, 6 (2014).

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach.-Learn. Res.* 3 (2003), 993–1022.

[3] Yanhua Chen, Manjeet Rege, Ming Dong, and Jing Hua. 2008. Non-negative matrix factorization for semi-supervised data clustering. *Knowl. Info. Syst.* 17, 3 (2008), 355–379.

[4] Philipp Cimiano, Michael Erdmann, and Günter Ladwig. 2007. Corpus-based pattern induction for a knowledge-based question answering approach. In *Proceedings of the IEEE International Conference on Semantic Computing (ICSC'07)*. IEEE, 671–678.

[5] Peter Clark, John Thompson, and Bruce Porter. 1999. A knowledge-based approach to question-answering. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'99)*, Vol. 99. Citeseer, 43–51.

[6] Oracle Corporation. [n.d.]. The Java Language Specification, Java SE 8th Edition. Retrieved from http://docs.oracle.com/javase/specs/jls/se8/html/index.html.
[7] Oracle Corporation. [n.d.]. Java Platform, Standard Edition 8 API Specification. Retrieved from https://docs.oracle.com/javase/8/docs/api/index.html.
[8] Daniel Hasan Dalip, Marcos André Gonçalves, Marco Cristo, and Pavel Calado. 2013. Exploiting user feedback to learn to rank answers in Q&A forums: A case study with stack overflow. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 543–552.
[9] Inderjit S. Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'01)*. ACM, 269–274. DOI: https://doi.org/10.1145/502512.502550
[10] Stack Exchange. [n.d.]. Stack Exchange Data Dump. Retrieved from https://archive.org/details/stackexchange.
[11] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In *Proceedings of the Knowledge Discovery and Data Mining (KDD'14)*. ACM, 1156–1165.
[12] Django Software Foundation. [n.d.]. Django Documentation. Retrieved from https://docs.djangoproject.com/en/1.9/.
[13] Python Software Foundation. [n.d.]. The Python Language Reference. Retrieved from https://docs.python.org/3/reference/index.html.
[14] Python Software Foundation. [n.d.]. The Python Standard Library. Retrieved from https://docs.python.org/3/library/index.html.
[15] The Mozilla Foundation. [n.d.]. JavaScript Reference. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference.
[16] The PHP Group. [n.d.]. PHP Documentation, PHP 7. Retrieved from http://php.net/docs.php.
[17] Ulf Hermjakob, Eduard H. Hovy, and Chin-Yew Lin. 2002. Knowledge-based question answering. In *Proceedings of the SCI Conference*.
[18] Google Incorporated. [n.d.]. Android Reference. Retrieved from http://developer.android.com/reference/packages.html.
[19] Jin Liu, Pingyi Zhou, Zijiang Yang, Xiao Liu, and John Grundy. 2018. FastTagRec: Fast tag recommendation for software information sites. *Auto. Softw. Eng.* (2018), 1–27.
[20] Stefania Mariano and Andrea Casey. 2007. The process of knowledge retrieval: A case study of an American high-technology research, engineering and consulting company. *VINE* 37, 3 (2007), 314–330.
[21] Avigit K. Saha, Ripon K. Saha, and Kevin A. Schneider. 2013. A discriminative model approach for suggesting tags automatically for stack overflow questions. In *Proceedings of the Mining Software Repositories Conference (MSR'13)*. IEEE Press, 73–76.
[22] A. K. Singh, N. K. Nagwani, and S. Pandey. 2017. TAGme: A topical folksonomy based collaborative filtering for tag recommendation in community sites. In *Proceedings of the 4th Multidisciplinary International Social Networks Conference*. ACM, 27.
[23] Parikshit Sondhi and ChengXiang Zhai. 2014. Mining semi-structured online knowledge bases to answer natural language questions on community QA websites. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM'14)*. ACM, 341–350.
[24] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *Proceedings of the World Wide Web Conference (WWW'16)*. 771–782.
[25] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. 2014. Knowledge base completion via search-based question answering. In *Proceedings of the World Wide Web Conference (WWW'14)*. ACM, 515–526.
[26] Yiyu Yao, Yi Zeng, Ning Zhong, and Xiangji Huang. 2007. Knowledge retrieval (KR). In *Proceedings of the Web Intelligence Consortium (WIC'07)*. IEEE, 729–735.
[27] Zhiping Zheng. 2003. Question answering using web news as knowledge base. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL'03)*. ACL, 251–254. DOI: https://doi.org/10.3115/1067737.1067797
[28] Zhou Zhibin, Shi Shuicai, Li Yuqin, and Lv Xueqiang. 2010. An answer extraction method of simple question based on web knowledge library. In *Proceedings of the Workshop on Education Technology and Computer Science (ETCS'10)*, Vol. 1. IEEE, 308–311.
[29] P. Zhou, J. Liu, Z. Yang, and G. Zhou. 2017. Scalable tag recommendation for software information sites. In *Proceedings of the IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*. 272–282.

Received November 2017; revised January 2019; accepted March 2019