# MATAR: Keywords Enhanced Multi-Label Learning for Tag Recommendation

Licheng Li, Yuan Yao, Feng Xu, and Jian Lu

State Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210046, China
{jimmyli,yyao}@smail.nju.edu.cn, {xf,lj}@nju.edu.cn

**Abstract.** Tagging is a popular way to categorize and search online content, and tag recommendation has been widely studied to better support automatic tagging. In this work, we focus on recommending tags for content-based applications such as blogs and question-answering sites. Our key observation is that many tags actually have appeared in the content in these applications. Based on this observation, we first model the tag recommendation problem as a multi-label learning problem and then further incorporate keyword extraction to improve recommendation accuracy. Moreover, we speedup the proposed method using a locality-sensitive hashing strategy. Experimental evaluations on two real data sets demonstrate the effectiveness and efficiency of our proposed methods.

**Keywords:** Tag recommendation, multi-label learning, keyword extraction, locality-sensitive hashing

## 1 Introduction

Tags have been widely used to categorize and search online content in many online applications such as blogs, question-answering sites, and online newspapers. For example, the question-answering site Stack Overflow[1] uses tags to categorize programming questions so that the questions can be easily found by appropriate answerers. Although tags are important for these applications, it also brings additional burdens to users. For example, when a user posts a programming question in Stack Overflow, he/she is required to add several appropriate tags for this question, which is not usually an easy task. This motivates the research to automatically recommend appropriate tags for users in these online applications.

Roughly, existing tag recommendation methods can be classified into two categories: collaborative-filtering method and content-based method (see Section 2 for a review). Methods in the first category rely on users' historical behavior, and these methods are more suitable to tag a relatively fixed set of items (e.g., music and movies). On the other hand, methods in the second category mainly use the

[1] http://stackoverflow.com/

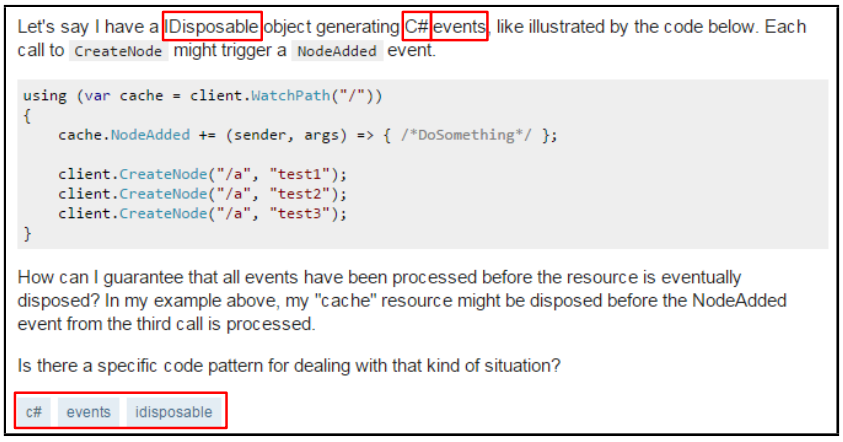



**Fig. 1.** A programming question from Stack Overflow.

content information, and they are more suitable for content-based applications (e.g., blogs and question-answering sites).

In this work, we focus on recommending tags for content-based applications. Our key observation is that in content-based applications, many tags actually have appeared in the content, and these tags are potential keywords in the content. Fig. 1 shows an illustrative example from Stack Overflow. There are three tags (i.e., 'c#', 'events', and 'idisposable') for the given question in this example. They all appeared in the question content, and they all revealed key information of the question. Actually, we empirically found that around 70% tags have appeared in the corresponding questions on Stack Overflow. Based on this observation, we aim to boost the accuracy of tag recommendation by combining two content-based methods: multi-label learning and keyword extraction. Specially, we first model the tag recommendation problem as a multi-label learning problem, and then incorporate a keyword extraction method into multi-label learning. Further, since one of the challenges of tag recommendation is from the large scale of the content data, we also speedup the proposed method by employing the locality-sensitive hashing strategy.

The main contributions of this paper are summarized as follows:

- We propose a tag recommendation method (MATAR) for improving recommendation accuracy in content-based applications. The proposed MATAR method combines the multi-label learning method and the keyword extraction method.
- We propose a fast version of MATAR, which employs approximation methods to reduce the time complexity of MATAR.
- Experimental results on two real data sets show that the proposed MATAR method outperforms several existing tag recommendation methods. In particular, MATAR improves the best competitor by 3.7% - 18.9% in terms

of recommendation accuracy, and the fast version of MATAR can handle a data set with more than 1 million posts with linear scalability.

The rest of the paper is organized as follows. We first review related work in Section 2. Then, we present the proposed MATAR algorithm together with its fast version in Section 3. We present the experimental results in Section 4 and conclude this paper in Section 5.

## 2 Related Work

In this section, we briefly review related work.

We roughly divide existing tag recommendation methods into two categories: *collaborative-filtering* method and *content-based* method.

For collaborative-filtering method, the key insight is to employ the tagging histories (i.e., user-item-tag tuples) from all users. For example, Symeonidis et al. [18] adopt tensor factorization model on the user-item-tag tuples; Rendle et al. [11, 12] further model the pairwise rankings into tensor factorization method. Other examples in this category include [4, 7, 15, 3]. Overall, methods in this category are more suitable to tag a relatively fixed set of items (e.g., music and movies), and they are not able to recommend tags for new content that has not been tagged yet.

In the second category of content-based method, the content itself is used as input. For example, some studies focus on the feature aspect by finding useful textual features (such as tag co-occurrence [2, 21] and entropy [8]). In the algorithm aspect, classification models and topic models are widely used for tag recommendation. For example, Saha et al. [13] train a classification model for each tag and recommend tags based on multiple classifiers; Krestel et al. [5] use topic models to find latent topics from the content and then recommend tags based on these latent topics; Si and Sun [14] further propose a variant LDA model to link tags with the latent topics. Other content-based methods include [16, 17, 20]. In this work, we also focus on content-based methods as we target at recommending tags for content-based applications.

There are also some other lines of research that are potentially useful for tag recommendation. Tag recommendation can be viewed as a multi-label learning problem by treating tags as labels. For example, TagCombine [21] is one of the few methods that applies multi-label learning for tag recommendation. However, TagCombine suffers from the class imbalance problem (i.e., each tag is only used by a very small portion of posts). Keyword extraction is another potentially useful tool for tag recommendation, as tags may appear as keywords in the content. For example, keywords may be directly used as tags [10]; association rules between keywords and tags are also considered [19]. In this work, we propose to incorporate keyword extraction into multi-label learning for better tag recommendation. To the best of our knowledge, we are the first to combine multi-label learning and keyword extraction for tag recommendation in content-based applications.

## 3 The Proposed Methods

In this section, we present our algorithms for tag recommendation. The proposed MATAR algorithm combines multi-label learning with keyword extraction to enhance the recommendation accuracy. We further propose MATAR-fast to speedup the computations of MATAR.

### 3.1 Problem Statement

Before presenting our algorithms, we first introduce some notations and define the tag recommendation problem. We use $\mathbf{X}_{m\times d}$ to denote the feature matrix where each row contains the features for an instance. Without loss of generality, we assume that there are $m$ instances and $d$ features for each instance. We use $\mathbf{Y}_{m\times T}$ to denote the associate tag matrix where $T$ is the number of tags. We use subscripts to indicate the entries. That is, $\mathbf{X}_i$ and $\mathbf{Y}_i$ indicate the feature vector and the tag vector of the $i^{th}$ instance, respectively. Then, element $\mathbf{Y}_i(t) \in \{0, 1\}$ indicates whether the $i^{th}$ instance is assigned with the $t^{th}$ tag ($\mathbf{Y}_i(t) = 1$ indicates that the instance is assigned with the $t^{th}$ tag and $\mathbf{Y}_i(t) = 0$ indicates otherwise). Based on the above notations, the tag recommendation takes the existing $\mathbf{X}$ matrix and $\mathbf{Y}$ matrix as input, and aims to predict the tag vector $\mathbf{Y}_r$ for a given new instance $\mathbf{X}_r$.

### 3.2 The Proposed MATAR Algorithm

Next, we describe the MATAR algorithm. To make the descriptions of our algorithms easier, we further define the following notations. For a given instance $\mathbf{X}_i$, we define $\mathcal{N}(\mathbf{X}_i)$ as the set of its $K$-nearest neighbors. Based on the tag vectors of these neighbors, we define a *counting* vector for $\mathbf{X}_i$ as

$$S_i(t) = \sum_{\mathbf{X}_j \in \mathcal{N}(\mathbf{X}_i)} \mathbf{Y}_j(t), t = 1, 2, \ldots, T. \tag{1}$$

Basically, $S_i(t)$ counts the number of instances that have been assigned with the $t^{th}$ tag among the neighborhood of $\mathbf{X}_i$.

Next, for a given new instance $\mathbf{X}_r$, we define two families of probability events. The first one is $U_g^t$ ($g \in \{0, 1\}$), where $U_1^t$ is the event that tag $t$ belongs to $\mathbf{X}_r$ and $U_0^t$ is the event that tag $t$ does not belong to $\mathbf{X}_r$. The other one is $V_k^t$ ($k \in \{0, 1, \ldots, K\}$), which means that there are exactly $k$ instances that have tag $t$ among the $K$-nearest neighbors of $\mathbf{X}_r$. Based on these two families of probability events, we aim to estimate the probability $P_r(t)$ by which the new instance $\mathbf{X}_r$ would be tagged with tag $t$ (i.e., the $t^{th}$ tag). Following the $K$-nearest neighborhood multi-label learning framework [22], we can define $P_r(t)$ as

$$P_r(t) = P(U_1^t | V_{S_r(t)}^t), t = 1, 2, \ldots, T. \tag{2}$$

In this work, we use the cosine similarity (i.e., the cosine distance between $\mathbf{X}_r$ and $\mathbf{X}_i$) to find the neighborhood.

Applying the Bayesian rule and decomposing the $V^t_{S_r(t)}$ event, we can rewrite Eq. (2) as

$$P_r(t) = \frac{P(U^t_1)P(V^t_{S_r(t)}|U^t_1)}{P(V^t_{S_r(t)})} = \frac{P(U^t_1)P(V^t_{S_r(t)}|U^t_1)}{\sum_{g\in\{0,1\}} P(U^t_g)P(V^t_{S_r(t)}|U^t_g)}. \tag{3}$$

**Computing $P(U^t_g)$ and $P(V^t_k|U^t_g)$.** As shown in Eq. (3), we need to estimate the prior $P(U^t_g)$ ($t = 1, 2, \ldots, T$ and $g \in \{0, 1\}$) and the posterior $P(V^t_k|U^t_g)$ ($k \in \{0, 1, \ldots, K\}$) to compute the probability $P_r(t)$. We resort to keyword extraction and frequency counting to compute these prior and posterior probabilities.

As mentioned in introduction, the key intuition of our method is that many tags actually appeared as keywords in the content. Let $w^t$ be the corresponding word of tag $t$, we have

$$P(U^t_g) = P(U^t_g|w^t)P(w^t) + P(U^t_g|\neg w^t)P(\neg w^t), \tag{4}$$

where $P(w^t)$ is probability of which the word $w^t$ is a keyword in the content, and $P(\neg w^t)$ is probability of which the word $w^t$ is not a keyword in the content. Here, $P(\neg w^t) = 1 - P(w^t)$.

Then, computing $P(U^t_g)$ becomes computing $P(U^t_g|w^t)$, $P(U^t_g|\neg w^t)$ and $P(w^t)$. For $P(w^t)$, various keyword extraction methods can be used here. In this work, we adopt the TextRank [9] method. The basic idea of TextRank is to model the text as a weighted graph. Each vertex in this graph represents a unique word in the text, and an edge between two words indicates that these two words are close to each other in the text. Then, pagerank algorithm is applied on this graph to find the keywords. The output of TextRank is a score vector in which each word is assigned with a score to indicate the importance of this word. In other words, the score for each word indicates the probability by which this word could be a keyword. Therefore, we use this score as $P(w^t)$.

To estimate $P(U^t_g|w^t)$, we can approximate the result by doing some statistics on the training data. For each tag $t$, we first define two variables $c_w$ and $c_{w,t}$. We first find the instances where the corresponding word of tag $t$ appears. Then, $c_w$ counts the number of these instances and $c_{w,t}$ counts the number of these instances that are tagged with $t$. Based on these two variables, we can estimate probability $P(U^t_g|w^t)$ as

$$\begin{aligned} P(U^t_1|w^t) &= \frac{c_{w,t} + s}{c_w + s \times 2} \\ P(U^t_0|w^t) &= 1 - P(U^t_1|w^t), \end{aligned} \tag{5}$$

where $s = 1$ is a smoothing parameter.

Similarly, to estimate $P(U^t_g|\neg w^t)$, we also define two variables $c_{\neg w}$ and $c_{\neg w,t}$ for each tag $t$. We first find the instances where the corresponding word of tag $t$ does not appear, and then $c_{\neg w}$ counts the number of these instances and $c_{\neg w,t}$

**Algorithm 1** The Training Stage of Our MATAR Algorithm

**Input:** $\mathbf{X}$, $\mathbf{Y}$, and $K$
**Output:** $P(U_g^t)$, $P(V_k^t|U_g^t)$, $P(U_g^t|w^t)$, and $P(U_g^t|\neg w^t)$
1: **for** $i \in \{1, 2, \ldots, m\}$ **do**
2: Compute $P(w^t)$ for each tag in $\mathbf{X}_i$;
3: Identify the neighborhood $\mathcal{N}(\mathbf{X}_i)$ for $\mathbf{X}_i$;
4: **end for**
5: **for** $t \in \{1, 2, \ldots, T\}$ **do**
6: Compute $P(U_g^t|w^t)$ and $P(U_g^t|\neg w^t)$ using Eq. (5) and Eq. (6);
7: Compute $P(U_g^t)$ using Eq. (7);
8: Compute $P(V_k^t|U_g^t)$ using Eq. (8);
9: **end for**

counts the number of these instances that are tagged with $t$. Based on these two variables, we can estimate probability $P(U_g^t|\neg w^t)$ as

$$\begin{aligned} P(U_1^t|\neg w^t) &= \frac{c_{\neg w,t} + s}{c_{\neg w} + s \times 2} \\ P(U_0^t|\neg w^t) &= 1 - P(U_1^t|\neg w^t). \end{aligned} \tag{6}$$

Notice that our computations of $P(U_g^t|w^t)$, $P(U_g^t|\neg w^t)$ and $P(w^t)$ are all based on the cases when the tags appeared in the content. If the tags did not appear in the content, we may directly estimate $P(U_g^t)$ as

$$\begin{aligned} P(U_1^t) &= \frac{\sum_{i=1}^{m} \mathbf{Y}_i(t) + s}{m + s \times 2} \\ P(U_0^t) &= 1 - P(U_1^t). \end{aligned} \tag{7}$$

Next, we show how we compute $P(V_k^t|U_g^t)$. For each tag $t$, we first define two arrays $c$ and $c'$ of length $K + 1$. We first find the training instances whose $K$-nearest neighbors contain exactly $k$ instances with tag $t$, and then $c[k]$ and $c'[k]$ count the number of these training instances that are tagged with $t$ and without $t$, respectively. Based on these two arrays, we can estimate the posterior probabilities as

$$\begin{aligned} P(V_k^t|U_1^t) &= \frac{c[k] + s}{\sum_{j=0}^{K} c[j] + s \times (K + 1)} \\ P(V_k^t|U_0^t) &= \frac{c'[k] + s}{\sum_{j=0}^{K} c'[j] + s \times (K + 1)}. \end{aligned} \tag{8}$$

The overall MATAR algorithm is summarized in Alg. 1 and Alg. 2 (training stage and predicting stage, respectively). Steps 1-4 in Alg. 1 compute $P(w^t)$ and $\mathcal{N}(\mathbf{X}_i)$ for each instance. Steps 5-9 estimate $P(U_g^t|w^t)$, $P(U_g^t|\neg w^t)$, $P(U_g^t)$ and $P(V_k^t|U_g^t)$. Based on the results of the training stage, the predicting stage is shown in Alg. 2. After computing $\mathcal{N}(\mathbf{X}_r)$ and $P(w^t)$ for $\mathbf{X}_r$, Steps 3-9 compute

**Algorithm 2** The Predicting Stage of Our MATAR Algorithm

**Input:** $\mathbf{X}$, $\mathbf{Y}$, $K$, $P(U_g^t)$, $P(V_k^t|U_g^t)$, $P(U_g^t|w^t)$, $P(U_g^t|\neg w^t)$, and $\mathbf{X}_r$
**Output:** $P_r(t)$
1: Identify $\mathcal{N}(\mathbf{X}_r)$;
2: Compute $P(w^t)$ for $\mathbf{X}_r$;
3: **for** $t \in \{1, 2, \ldots, T\}$ **do**
4: Compute $S_r(t)$ using Eq. (1);
5: **if** $P(w^t) \neq 0$ **then**
6: Update $P(U_g^t)$ using Eq. (4);
7: **end if**
8: Compute $P_r(t)$ using Eq. (3);
9: **end for**

the probability $P_r(t)$ for each possible tag. During the iterations, we update $P(U_g^t)$ if $P(w^t) \neq 0$ and use the input $P(U_g^t)$ otherwise. Finally, we can recommend a ranking list based on the $P_r(t)$ scores for each possible tag.

**Algorithm Analysis.** Here, we present some algorithm analysis for MATAR. Iterations of Step 2 in Alg. 1 require $O(md)$ time. The main time consumption is from Step 3. Iterations of this step require $O(m^2d)$ time. From Step 5 to Step 9, we need $O(mTK)$ time. In general, the time complexity of the training stage of our MATAR algorithm is $O(m^2d+mTK)$. Since $K$ is usually much smaller than $m$ and $T$, and the feature dimension $d$ is a fixed constant, the time complexity of the training stage can be rewritten as $O(m^2 + mT)$.

For Alg. 2, Step 1 requires $O(md)$ time. Step 2 requires $O(d)$ time. Steps 3-9 compute the probabilities $P_r(t)$, which requires $O(TK)$ time. To sum up, the time complexity of the predicting stage of our MATAR algorithm is $O(md+TK)$, which can be rewritten as $O(m + T)$.

### 3.3 The Proposed MATAR-fast Algorithm

Next, we present a fast version of MATAR. The main time consumption of our MATAR algorithm is from the nearest neighbor computations. That is, we need to compute the similarities between any two instances. To speedup MATAR, we employ an approximate method to linearly identify neighbors. Specially, we use locality-sensitive hashing [1] whose basic idea is to hash the input instances so that similar instances are mapped to the same buckets with high probability (the number of buckets is usually much smaller than the number of input instances). Therefore, we only need to compute the similarities within each bucket.

We consider the family of hash functions defined as follows. For each hash function in this family, we first choose a random vector $\boldsymbol{v}_{d\times 1}$ where $d$ is the dimension size, and then define the following hash function $h_{\boldsymbol{v}}$:

$$h_{\boldsymbol{v}}(\boldsymbol{a}) = \begin{cases} 1 & \text{if } \boldsymbol{v} \cdot \boldsymbol{a} \geq 0 \\ 0 & \text{if } \boldsymbol{v} \cdot \boldsymbol{a} < 0 \end{cases} \tag{9}$$

**Table 1.** The statistics of SO, SO-Small and Math data sets.

| Data | # Posts | # Tags | # Tag Assignments |
|---|---|---|---|
| SO | 1,942,249 | 777 | 4,358,738 |
| SO-Small | 10,000 | 777 | 22,335 |
| Math | 19,953 | 501 | 36,813 |

Further, we randomly pick $L$ hash functions from this family where $L$ determines the number of buckets. Applying these $L$ hash functions to one input instance would output a bit vector composed of 1s or 0s (the bit vector is regarded as the encoding of a bucket). Then, we can compute the similarities within each bucket as an approximation for similarity computations. The detailed algorithm is omitted for simplicity.

**Algorithm Analysis.** Finally, we analyze the complexity of MATAR-fast. As we stated above, the locality-sensitive hashing strategy is applied to find the neighborhood in Alg. 1 to reduce the complexity. The time complexity of MATAR-fast mainly depends on the $L$ parameter, which is $O(\frac{m^2}{2^L})$. Typically, we can set $L = O(\log m)$ (e.g., let $L = \log \frac{m}{500}$, there are averagely 500 instances in each bucket). Therefore, the similarity computation for each $\mathbf{X}_i$ requires $O(1)$ time as it only involves the instances in the same bucket. Overall, the time complexity of the training stage of our MATAR-fast algorithm is $O(md + mTK)$, which can be rewritten as $O(mT)$.

## 4 Experiments

In this section, we present the experimental evaluations. The experiments are designed to answer the following questions:

- *Effectiveness*: How accurate are the proposed algorithms for recommending tags?
- *Efficiency*: How scalable are the proposed algorithms?

### 4.1 Experiment Setup

**Data Sets.** We use the data from two real world sites, i.e., Stack Overflow (SO) and Mathematics Stack Exchange (Math), to evaluate our algorithms. They are popular CQA sites for programming and math, respectively. For both data sets, they are officially published and publicly available[2]. For features, we adopt the commonly used 'bag of words' model. For tags, we put our focus on common tags and remove some rare tags. To compare with the existing methods, we also randomly select a small subset of the SO data. The statistics of the three data sets are summarized in Table 1.

[2] http://blog.stackoverflow.com/category/cc-wiki-dump/

**Table 2.** The effectiveness comparisons. Higher is better. The proposed algorithms (MATAR and MATAR-fast) outperform all the other methods.

(a) Result on Math data

| Method | Recall@5 | Recall@10 |
|---|---|---|
| SVMSim | 0.5903 | 0.6770 |
| MLKNN | 0.5247 | 0.6267 |
| LDASim | 0.5139 | 0.6191 |
| TagCombine | 0.5908 | 0.6429 |
| Snaff | 0.2904 | 0.3671 |
| **MATAR** | **0.6123** | **0.7122** |
| **MATAR-fast** | **0.5941** | **0.6965** |

(b) Result on SO-Small data

| Method | Recall@5 | Recall@10 |
|---|---|---|
| SVMSim | 0.3219 | 0.3803 |
| MLKNN | 0.3563 | 0.4476 |
| LDASim | 0.3447 | 0.4402 |
| TagCombine | 0.4490 | 0.4936 |
| Snaff | 0.2148 | 0.2586 |
| **MATAR** | **0.4758** | **0.5871** |
| **MATAR-fast** | **0.4504** | **0.5531** |

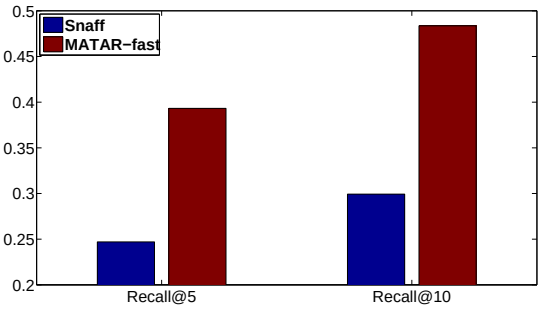



**Fig. 2.** The result on the entire SO data. Higher is better. Our MATAR-fast is much better than Snaff.

**Evaluation Metrics.** For effectiveness evaluation, recall is more important than precision in our problem setting. Therefore, we adopt the recall@k metric as defined below.

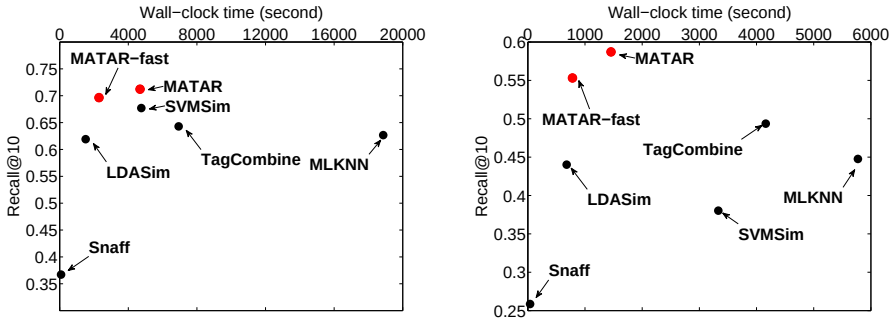
$$Recall@k = \frac{1}{m}\sum_{i=1}^{m}\frac{|Rec_i \cap Tag_i|}{|Tag_i|}, \tag{10}$$

where $m$ is the number of test instances, $Tag_i$ is the actual tags for instance $i$, $Rec_i$ is the top-k ranked tags recommended for instance $i$.

For efficiency, we report the wall-clock time. All the efficiency experiments were run on a machine with eight 3.4GHz Intel Cores and 32GB memory.

### 4.2 Experimental Results

**Effectiveness Results.** We first compare the effectiveness of the proposed algorithms (MATAR and MATAR-fast) with SVMSim [13], TagCombine [21], LDASim [5], MLKNN [22], and Snaff [6] on Math data and SO-Small data. We randomly choose 90% data for training and use the rest 10% data for testing. We set $K = 60$ for Math data and $K = 70$ for SO-Small data. The results on the two data sets are shown in Table 2.




(a) Tag Recommendation on Math data (b) Tag Recommendation on SO data

**Fig. 3.** The quality-speed tradeoff. The proposed MATAR-fast achieves a good balance between the recommendation accuracy and the efficiency (in the left-top corner).

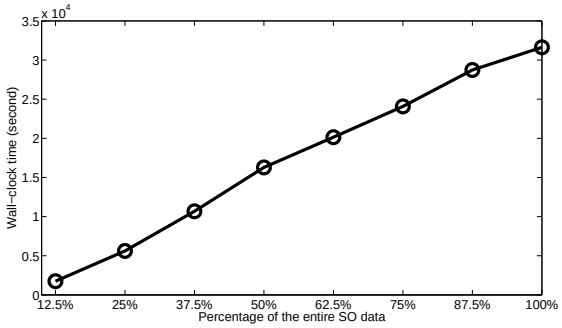



**Fig. 4.** Scalability of MATAR-fast. Our MATAR-fast scales linearly wrt the number of instances.

We make several observations from Table 2. First, the proposed MATAR algorithm performs the best. For example, on Math data, MATAR improves the best competitor (i.e., SVMSim) by 3.7% and 5.2% for recall@5 and recall@10, respectively; on the SO-Small data, it improves the best competitor (i.e., TagCombine) by 6.0% and 18.9% for recall@5 and recall@10, respectively. Second, MATAR also performs much better than MLKNN. For example, on Math data, MATAR improves the MLKNN method by 16.7% and 13.6% for recall@5 and recall@10, respectively. Since MATAR can be seen as an extension of MLKNN, this result indicates that the keyword extraction indeed helps in tag recommendation. Third, although the performance of the proposed MATAR-fast algorithm is not as good as MATAR, it still better than the compared methods.

We also give the results on the entire SO data in Fig. 2. Here, we only show the results of MATAR-fast and Snaff as only these two algorithms can return results within 24 hours on the entire SO data. As we can see from the figure, our

MATAR-fast performs much better than Snaff, i.e., it improves Snaff by 59.2% and 61.6% for recall@5 and recall@10, respectively.

**Quality-Speed Tradeoff.** Next, we study the quality-speed tradeoff of different algorithms in Fig. 3. In the figure, we plot the recall@10 on the y-axis and the wall-clock time on the x-axis. Ideally, we want an algorithm sitting in the left-top corner. As we can see, our MATAR and MATAR-fast are both in the left-top corner. For example, compared with TagCombine, MATAR-fast is 5.3x faster wrt wall-clock time and 18.9% better wrt recall@10 on the SO-Small data. Although Snaff is faster than our methods, the accuracy of this method is much worse. Overall, our MATAR-fast achieves a good balance between the recommendation accuracy and the efficiency.

**Scalability Results.** Finally, we study the scalability of our proposed MATAR-fast algorithm on the whole SO data. We vary the size of training data, and report the wall-clock time in Fig. 4. As we can see from the figure, the running time of our MATAR-fast algorithm scales linearly wrt the number of training data size, which is also consistent with our algorithm analysis in Section 3.3.

## 5 Conclusions

In this paper, we have proposed a novel tag recommendation algorithm MATAR. The proposed MATAR combines multi-label learning and keyword extraction to enhance the recommendation accuracy. The basic intuition behind MATAR is the fact that many tags actually have appeared in the content in content-based applications. We have also proposed a fast version for MATAR based on the locality-sensitive hashing strategy. Experimental evaluations on two real data sets demonstrate the effectiveness and efficiency of the proposed methods.

## References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on. pp. 459–468. IEEE (2006)
2. Belém, F., Martins, E., Pontes, T., Almeida, J., Gonçalves, M.: Associative tag recommendation exploiting multiple textual features. In: Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval. pp. 1033–1042. ACM (2011)
3. Feng, W., Wang, J.: Incorporating heterogeneous information for personalized tag recommendation in social tagging systems. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1276–1284. ACM (2012)
4. Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., Stumme, G.: Tag recommendations in social bookmarking systems. Ai Communication 21(4), 231–247 (2008)

5. Krestel, R., Fankhauser, P., Nejdl, W.: Latent dirichlet allocation for tag recommendation. In: Proceedings of the third ACM conference on Recommender systems. pp. 61–68. ACM (2009)
6. Lipczak, M., Milios, E.: Learning in efficient tag recommendation. In: Proceedings of the fourth ACM conference on Recommender systems. pp. 167–174. ACM (2010)
7. Liu, R., Niu, Z.: A collaborative filtering recommendation algorithm based on tag clustering. In: Future Information Technology, pp. 177–183. Springer (2014)
8. Lu, Y.T., Yu, S.I., Chang, T.C., Hsu, J.Y.j.: A content-based method to enhance tag recommendation. In: IJCAI. vol. 9, pp. 2064–2069 (2009)
9. Mihalcea, R., Tarau, P.: Textrank: Bringing order into texts. Association for Computational Linguistics (2004)
10. Murfi, H., Obermayer, K.: A two-level learning hierarchy of concept based keyword extraction for tag recommendations. ECML PKDD Discovery Challenge 2009 (DC09) p. 201
11. Rendle, S., Balby Marinho, L., Nanopoulos, A., Schmidt-Thieme, L.: Learning optimal ranking with tensor factorization for tag recommendation. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 727–736. ACM (2009)
12. Rendle, S., Schmidt-Thieme, L.: Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the third ACM international conference on Web search and data mining. pp. 81–90. ACM (2010)
13. Saha, A.K., Saha, R.K., Schneider, K.A.: A discriminative model approach for suggesting tags automatically for stack overflow questions. In: Proceedings of the 10th Working Conference on Mining Software Repositories. pp. 73–76. IEEE Press (2013)
14. Si, X., Sun, M.: Tag-lda for scalable real-time tag recommendation. Journal of Computational Information Systems 6(1), 23–31 (2009)
15. Sigurbjörnsson, B., Van Zwol, R.: Flickr tag recommendation based on collective knowledge. In: Proceedings of the 17th international conference on World Wide Web. pp. 327–336. ACM (2008)
16. Song, Y., Zhang, L., Giles, C.L.: Automatic tag recommendation algorithms for social recommender systems. ACM Transactions on the Web (TWEB) 5(1), 4 (2011)
17. Subramaniyaswamy, V., Pandian, S.C.: Topic ontology-based efficient tag recommendation approach for blogs. International Journal of Computational Science and Engineering 9(3), 177–187 (2014)
18. Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: Tag recommendations based on tensor dimensionality reduction. In: Proceedings of the 2008 ACM conference on Recommender systems. pp. 43–50. ACM (2008)
19. Wang, J., Hong, L., Davison, B.D.: Rsdc09: Tag recommendation using keywords and association rules. ECML PKDD discovery challenge pp. 261–274 (2009)
20. Wang, T., Wang, H., Yin, G., Ling, C.X., Li, X., Zou, P.: Tag recommendation for open source software. Frontiers of Computer Science 8(1), 69–82 (2014)
21. Xia, X., Lo, D., Wang, X., Zhou, B.: Tag recommendation in software information sites. In: Proceedings of the 10th Working Conference on Mining Software Repositories. pp. 287–296. IEEE Press (2013)
22. Zhang, M.L., Zhou, Z.H.: Ml-knn: A lazy learning approach to multi-label learning. Pattern recognition 40(7), 2038–2048 (2007)