Zhang Y, Lo D, Xia X *et al.* Multi-factor duplicate question detection in Stack Overflow. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(5): 981–997 Sept. 2015. DOI 10.1007/s11390-015-1576-4

Multi-Factor Duplicate Question Detection in Stack Overflow

Yun Zhang [1] (张 芸), David Lo [2], *Member, ACM, IEEE*, Xin Xia [1,*] (夏 鑫), *Member, CCF, ACM, IEEE* and Jian-Ling Sun [1] (孙建伶), *Member, CCF, ACM*

[1] *College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*
[2] *School of Information Systems, Singapore Management University, Singapore, Singapore*

E-mail: yunzhang28@zju.edu.cn; davidlo@smu.edu.sg; {xxia, sunjl}@zju.edu.cn

Received July 15, 2014; revised July 17, 2015.

Abstract Stack Overflow is a popular on-line question and answer site for software developers to share their experience and expertise. Among the numerous questions posted in Stack Overflow, two or more of them may express the same point and thus are duplicates of one another. Duplicate questions make Stack Overflow site maintenance harder, waste resources that could have been used to answer other questions, and cause developers to unnecessarily wait for answers that are already available. To reduce the problem of duplicate questions, Stack Overflow allows questions to be manually marked as duplicates of others. Since there are thousands of questions submitted to Stack Overflow every day, manually identifying duplicate questions is a difficult work. Thus, there is a need for an automated approach that can help in detecting these duplicate questions. To address the above-mentioned need, in this paper, we propose an automated approach named DupPredictor that takes a new question as input and detects potential duplicates of this question by considering multiple factors. DupPredictor extracts the title and description of a question and also tags that are attached to the question. These pieces of information (title, description, and a few tags) are mandatory information that a user needs to input when posting a question. DupPredictor then computes the latent topics of each question by using a topic model. Next, for each pair of questions, it computes four similarity scores by comparing their titles, descriptions, latent topics, and tags. These four similarity scores are finally combined together to result in a new similarity score that comprehensively considers the multiple factors. To examine the benefit of DupPredictor, we perform an experiment on a Stack Overflow dataset which contains a total of more than two million questions. The result shows that DupPredictor can achieve a recall-rate@20 score of 63.8%. We compare our approach with the standard search engine of Stack Overflow, and DupPredictor improves its recall-rate@10 score by 40.63%. We also compare our approach with approaches that only use title, description, topic, and tag similarity and Runeson *et al.*'s approach that has been used to detect duplicate bug reports, and DupPredictor improves their recall-rate@10 scores by 27.2%, 97.4%, 746.0%, 231.1%, and 16.4% respectively.

Keywords software information site, duplicate question, Stack Overflow, DupPredictor

1 Introduction

Nowadays, software engineers use various software information sites to search, communicate, collaborate, and share information with one another[1]. Software information sites play an important role in the whole life cycle of software engineering[2-3]. Stack Overflow is one of the most popular software information sites where people ask and answer technical questions about software development and maintenance. In November 2014, Stack Overflow contained more than eight million questions which cover a wide range of topics such as programming languages, software tool usage, and project management.

In Stack Overflow, some questions may describe the same problem, and we refer to them as duplicate ques-

Regular Paper
Special Section on Software Systems
This work was partially supported by the China Knowledge Centre for Engineering Sciences and Technology under Grant No. CKCEST-2014-1-5, the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grant Nos. 2015BAH17F01 and 2013BAH01B01, and the Fundamental Research Funds for the Central Universities of China.

*Corresponding Author

tions. For example, Fig.1 presents two duplicate questions which describe the same problem about finding a C++ OpenSource project for novice developers. Duplicate questions are raised by different users at different time points. In Stack Overflow, users could label a question as a duplicate and provide the ID of the question they believe to be duplicated①. In the group of duplicate questions, one of them is marked as "master" (i.e., the one marked as "[closed]" in Fig.1), while the others are marked as "duplicate". Unidentified duplicate questions increase the difficulty in the maintenance of Stack Overflow site, and waste valuable resources that are spent on the redundant effort of answering each of the questions separately. Furthermore, by not identifying duplicate questions as such, developers who ask duplicate questions potentially need to wait for a long time before their questions get answered, while ready answers are already available.

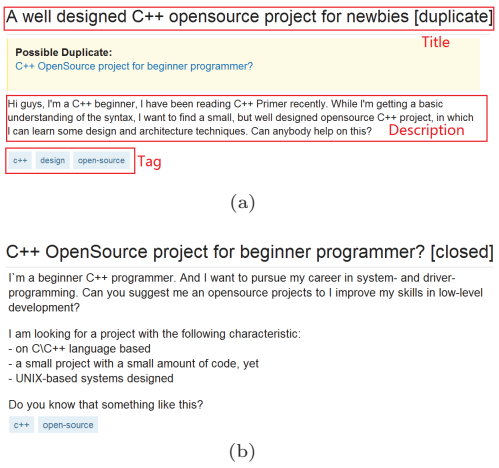


Fig.1. Two duplicate questions in Stack Overflow. (a) Question 3106628. (b) Question 634951.

The current state-of-the-practice is to identify these duplicate questions manually. Unfortunately, considering that there are thousands of questions submitted to Stack Overflow every day, manually identifying all duplicate questions would be a hard and tedious work. As a result, many duplicate questions are likely not identified as such and some questions might also be wrongly marked as duplicates②. Thus, there is a need for an automated tool which can be used to assist in the detection of duplicate questions. This tool can help save developers' and Stack Overflow site maintainers' time and improve the organization of the Stack Overflow site.

To address the above mentioned need, in this paper, we propose an automated approach named DupPredictor to detect duplicate questions by considering multiple factors. DupPredictor measures the similarity between two questions by using some observable factors, including titles, descriptions, and tags which can be directly extracted from the questions, and some latent factors corresponding to the topic distributions which are learned from the natural language descriptions of the questions. A topic modelling algorithm, namely Latent Dirichlet Allocation (LDA)[4], is used to infer the inherent topic distribution of a question. DupPredictor automatically combines these four factors by assigning different weights to the factors③. The following usage scenarios illustrate the benefits of our proposed tool.

Scenario 1 — *Without Tool.* Yun is a Java developer in a software company. One day, she encounters a problem with the Java compiler, so she posts a question in Stack Overflow. However, due to the large number of questions which are posted to Stack Overflow every day, other users do not pay attention to her question, and no one replies her question. About a month later, another enthusiastic developer David finds that the question is almost the same as one of the answered (aka., closed) questions he asked before. Thus David marks the question as a duplicate of a previous question. Finally, Yun gets the answer to her question, but it is one month later.

Scenario 2 — *With Tool.* Yun is a Java developer in a software company. One day, she encounters a problem with the Java compiler, so she posts a question in Stack Overflow. By using our automated tool which lists top 20 closed questions which may be duplicates of her question, Yun finds that one of the closed questions is similar to hers, so she checks the answer and finds the solution. The whole process completes in several minutes.

We evaluate our approach on the official dump of Stack Overflow data which was updated in August 2012[5]. In total, we extract more than two million questions, and among these questions, 1 528 questions are labeled as duplicates. We measure the performance of DupPredictor in terms of recall-rate@k. The experimental results show that our DupPredictor can achieve recall-rate@5, recall-rate@10 and recall-rate@20 values of 42.3%, 53.3% and 63.8%, respectively.

① http://meta.stackexchange.com/questions/10841/how-should-duplicate-questions-be-handled, July 2015.

② For more details, please refer to Subsection 5.4.

③ For more details, please refer to Section 3.

We compare DupPredictor with the standard search engine of Stack Overflow, four approaches which only consider one of the four factors (i.e., title, description, topic, and tags), and Runeson *et al.*'s approach[6] that was used to detect duplicate bug reports. The experimental results show that DupPredictor improves the standard search engine of Stack Overflow in terms of recall-rate@5 and recall-rate@10 by 13.71% and 40.63% respectively. And DupPredictor improves the recall-rate@5 of the four stand-alone approaches and Runeson *et al.*'s approach by 27.4%, 85.5%, 800.0%, 264.7%, and 10.2% respectively, the recall-rate@10 of these approaches by 27.2%, 97.4%, 746.0%, 231.1%, and 16.4% respectively, and the recall-rate@20 of these approaches by 26.0%, 93.9%, 717.9%, 225.5%, and 20.8% respectively.

The main contributions of this paper are as follows.

1) We propose the problem of duplicate question detection in Stack Overflow. We propose a novel approach named DupPredictor which considers and integrates multiple factors to detect duplicate questions.

2) We evaluate DupPredictor on more than two million questions in Stack Overflow. The experimental results show that DupPredictor can achieve a recall-rate@20 of 63.8% and substantially improves four baseline approaches.

The remainder of this paper is organized as follows. We elaborate the motivation of our work and introduce Latent Dirichlet Allocation (LDA) in Section 2. We describe the overall framework and the details of DupPredictor in Section 3. We present our experiments and their results in Section 4. Section 5 discuss some issues about the performance, efficiency, and threats to validity of DupPredictor. We review related work in Section 6. We conclude this paper and mention future work in Section 7.

2 Preliminaries

In this section, we first elaborate the motivation of our work. We then briefly introduce Latent Dirichlet Allocation (LDA), which is used to extract topic distributions from natural language descriptions of questions.

2.1 Motivation

Fig.1 presents two duplicate questions in Stack Overflow: question 3106628④ was labeled by a user to be a duplicate of question 634951⑤. A typical question in Stack Overflow contains a number of fields, such as submitter, title, description, tags, and comments. In this paper, we mainly consider three factors of the fields: title, description, and tags. A developer needs to provide all three pieces of information when he/she submits a question to Stack Overflow. The title is a summary of the question, the description is a detailed explanation of the question, and tags are sets of words or short phrases that capture important aspects of the question.

From Fig.1, we notice that the titles of the two questions are similar. They contain many common words, such as "C++", "opensource", and "project". Besides, both of the two questions are tagged with "C++" and "open-source". Moreover, although the words in the descriptions of the two questions are different, both of them describe a request for recommending a C++ open source project for novice developers. Thus, the latent meaning of these two questions is the same.

Observations and Implications. From the two duplicate questions, we have the following observations.

1) The title and tags are good factors to identify whether two questions are duplicates of each other. The two questions in Fig.1 share a number of common words in their titles and they also share a set of common tags.

2) The words used in the descriptions of two questions that are duplicates of each other may be different. However, the descriptions must share the same latent meaning. For example, the two questions in Fig.1 both describe the same request for recommending a C++ open source project for novice developers.

The second observation tells us that duplicate questions must share some latent commonalities, and these commonalities are also good factors to identify duplicate questions. A topic model such as Latent Dirichlet Allocation (LDA) can be used to infer these latent commonalities. LDA can be used to convert the natural language text contents in questions to their topic distributions. These topic distributions can then be compared to identify the latent commonalities among questions. In Subsection 2.2, we introduce LDA.

2.2 Latent Dirichlet Allocation

A topic model views a document to be a probability distribution of topics, while a topic is a probability distribution of words. In our setting, a document is

④http://stackoverflow.com/questions/3106628, July 2015.

⑤http://stackoverflow.com/questions/634951, July 2015.

the description of a question, and a topic is a higher-level concept corresponding to a distribution of words. For example, we can have a topic "Java Programming" which is a distribution of words such as "variable", "inheritance", "class", and "method".

Latent Dirichlet Allocation (LDA) is a well-known topic modeling technique proposed by Blei *et al.*[4] LDA is a generative probabilistic model of a textual corpus (i.e., a set of textual documents), which takes a training textual corpus as input, and a number of parameters including the number of topics (K) considered. In the training phase, for each document s, LDA will compute its topic distribution $\boldsymbol{\theta}_s$, which is a vector with K elements, and each element corresponds to a topic. The value of each element in $\boldsymbol{\theta}_s$ is a real number from 0 to 1, which represents the proportion of the words in s that belong to the corresponding topic in s. After training, LDA can be used to predict the topic distribution $\boldsymbol{\theta}_m$ of a new document m.

3 Our Proposed Approach

In this section, we first present the overall framework of our DupPredictor approach. Then we elaborate the details of the four components of DupPredictor: title similarity component, description similarity component, topic similarity component, and tag similarity component. Each of these components computes similarities of questions by considering one of the four factors. Finally, we describe how these four components are combined in DupPredictor.

3.1 Overall Framework

Fig.2 presents the overall framework of DupPredictor. The framework contains two phases: model building phase and prediction phase. In the model building phase, our goal is to train a model from historical duplicate questions which have been detected. In the prediction phase, this model would be used to detect new duplicate questions.

Our framework first collects historical questions from Stack Overflow. Then we preprocess the questions (step 1). In the preprocessing step, we first extract the title, the description, and tags from each question. Next, we tokenize the text that appears in the title and description of each question, remove common English stop words, and perform stemming. Stop words are commonly occurring words, e.g., "a", "the", "and", "he". Since they appear very often, they are of little help in differentiating different documents. Stemming is the process to reduce a word to its root form. For example, by using stemming, the words "marks", "marked", and "marking" are all reduced to "mark". We make use of the popular Porter stemming algorithm[7] which has been used in many other studies, e.g., [8-9].

After we have preprocessed the questions, for each question which is a duplicate of another question, we compare the question with all other questions that are submitted earlier. For each pair of questions, we compute four scores that capture the similarity of the questions. These scores are computed by the title similarity component, description similarity component, topic similarity component, and tag similarity component shown in Fig.2 (step 2)⑥. Next, we input these four sets of similarity scores into the composer component, which would then automatically learn a good weight for each of the four components (step 3)⑦.

After the composer component has learned the weights, in the prediction phase, it is used to return

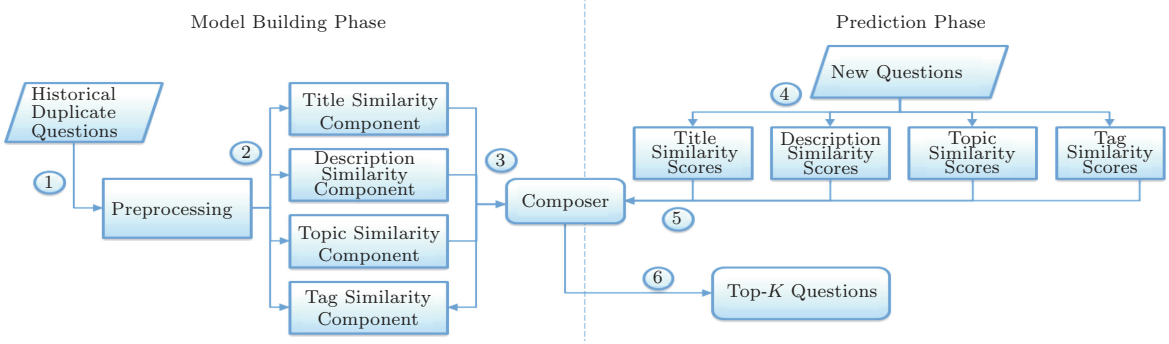


Fig.2. Overall framework of DupPredictor.

⑥For more details, please refer to Subsections 3.2, 3.3, 3.4, and 3.5, respectively.

⑦For more details, please refer to Subsection 3.6.

a list of K historical questions that are potentially duplicates of a new question. To get this top-K list, we compare every new question with all past questions. For each pair of questions, we compute their title similarity score, description similarity score, topic similarity score, and tag similarity score using the title, description, topic, and tag similarity component respectively (step 4). Next, we input these scores into the composer component which would then compute a final aggregate similarity score which is used to rank the existing questions (step 5). The top-K most similar questions are then outputted (step 6).

3.2 Title Similarity Component

This component computes the similarity between the titles of a pair of questions based on common words that they share. After preprocessing, the titles of two questions are transformed into two bags (i.e., multisets) of words. For two questions m and n, we represent the two bags of words that are extracted from their titles as $TitleBag_m$ and $TitleBag_n$ respectively. Next, we merge $TitleBag_m$ and $TitleBag_n$ and eliminate duplicate words to obtain the union set $TitleBag_u$, which contains v words. Following vector space modeling (VSM)[10], we represent the two titles as two vectors: $\boldsymbol{TitleVec}_m = (wt_{m,1}, wt_{m,2}, \ldots, wt_{m,v})$ and $\boldsymbol{TitleVec}_n = (wt_{n,1}, wt_{n,2}, \ldots, wt_{n,v})$. The weight $wt_{q,i}$ in these two vectors denotes the relative term frequency of the i-th word in question q's title, which is computed as follows:

$$wt_{q,i} = \frac{n_{q,i}}{\sum_v n_{q,v}}. \tag{1}$$

In (1), $n_{q,i}$ denotes the number of times the i-th word of $TitleBag_u$ appears in the title of question q, $\sum_v n_{q,v}$ denotes the total number of occurrences of all words in the title of question q, where v is the index of a word in $TitleBag_u$. We measure the similarity between two questions' titles by computing the cosine similarity[10] of their vector representations $\boldsymbol{TitleVec}_m$ and $\boldsymbol{TitleVec}_n$ as follows:

$$\begin{aligned} &TitleSim(\boldsymbol{TitleVec}_m, \boldsymbol{TitleVec}_n) \\ &= \frac{\boldsymbol{TitleVec}_m \cdot \boldsymbol{TitleVec}_n}{|\ \boldsymbol{TitleVec}_m\ ||\ \boldsymbol{TitleVec}_n\ |}. \end{aligned} \tag{2}$$

The numerator of (2) which is the dot product of the two vectors $\boldsymbol{TitleVec}_m = (wt_{m,1}, wt_{m,2}, \ldots, wt_{m,v})$ and $\boldsymbol{TitleVec}_n = (wt_{n,1}, wt_{n,2}, \ldots, wt_{n,v})$ is computed as follows:

$$\begin{aligned} &\boldsymbol{TitleVec}_m \cdot \boldsymbol{TitleVec}_n \\ &= wt_{m,1} \times wt_{n,1} + wt_{m,2} \times wt_{n,2} + \ldots + \\ &\quad wt_{m,v} \times wt_{n,v}. \end{aligned}$$

The terms $|\boldsymbol{TitleVec}_m|$ and $|\ \boldsymbol{TitleVec}_n|$ in the denominator of (2) denote the sizes of the two vectors respectively. The size of a vector $\boldsymbol{TitleVec}_m$ is computed as follows:

$$|\boldsymbol{TitleVec}_m| = \sqrt{wt_{m,1}^2 + wt_{m,2}^2 + \ldots + wt_{m,v}^2}.$$

For a new question nq and an old question oq, we denote their title similarity score computed using (2) as $TitleSim_{nq}(oq)$.

3.3 Description Similarity Component

This component calculates the similarity between the descriptions of a pair of questions based on common words that they share. After preprocessing, we transform the descriptions to bags of words. We denote two bags of words extracted from the descriptions of questions m and n as $DesBag_m$ and $DesBag_n$ respectively. Next, we merge $DesBag_m$ and $DesBag_n$ and eliminate duplicate words to obtain the union set $DesBag_u$, which contains a total of v words. Following vector space modeling[10], the two descriptions can be represented as two vectors: $\boldsymbol{DesVec}_m = (wd_{m,1}, wd_{m,2}, \ldots, wd_{m,v})$ and $\boldsymbol{DesVec}_n = (wd_{n,1}, wd_{n,2}, \ldots, wd_{n,v})$. The weight $wd_{q,i}$ in these two vectors denotes the term frequency of the i-th word in question q's description, which is computed as follows:

$$wd_{q,i} = \frac{n_{q,i}}{\sum_v n_{q,v}}. \tag{3}$$

In (3), $n_{q,i}$ denotes the number of times the i-th word of $DesBag_u$ appears in the description of question q, $\sum_v n_{q,v}$ denotes the total number of occurrences of all words in the description of question q, where v is the index of a word in $DesBag_u$. We measure the similarity between two questions' descriptions by computing the cosine similarity[10] of their representative vectors $\boldsymbol{DesVec}_m$ and $\boldsymbol{DesVec}_n$, similar to what is done to compute title similarity which is described in Subsection 3.2.

For a new question nq and an old question oq, we denote their description similarity score as $DesSim_{nq}(oq)$.

3.4 Topic Similarity Component

This component computes the similarity between the topic distributions of the textual contents (i.e., title + description) of a pair of questions. The topic distribution of a question captures the latent meaning of the question. We use a topic model, namely Latent Dirichlet Allocation (LDA) which is described in Subsection 2.2, to compute the topic distribution of a question. LDA represents a document as a probability distribution of topics, where a topic is a probability distribution of words. We input the stemmed non-stop word contents of the titles and descriptions of questions into LDA to extract the topic distribution of each question. Each topic distribution of a question is a vector with K elements, and each element corresponds to a topic. The value of each element is the proportion of words in the question that belong to the corresponding topic.

Consider a set of topic distributions T corresponding to the set of all questions. Let $\boldsymbol{T}_d = (p_{d,1}, p_{d,2}, \ldots, p_{d,t})$ refer to the topic distribution corresponding to question d, where $p_{d,j}$ denotes the probability of question d to belong to topic j. Here t is the topic number of the LDA model, which is determined by measuring the perplexity of the generated LDA model with t topics. Perplexity is a measure of the ability of a model to generalize to unseen data[11-12]. We test a set of topic numbers and choose the one with the best perplexity. We find that $t = 100$ gives the best perplexity. We measure the topic similarity of two questions by computing the cosine similarity[10] of their topic distributions. The cosine similarity of two topic distributions is computed in the same way as the cosine similarity of two vectors described in Subsection 3.2.

For a new question nq and an old question oq, we denote their topic similarity score as $TopicSim_{nq}(oq)$.

3.5 Tag Similarity Component

This component measures the similarity between the tags of a pair of questions. As tags summarize the pertinent aspects of a question from different perspectives and they can either appear or not appear in the question's title and description, we can gain additional useful information from the tags. For two questions m and n, we put their tags in $TagSet_m$ and $TagSet_n$ respectively. Next, we merge $TagSet_m$ and $TagSet_n$ and eliminate duplicate words, to obtain the union set $TagSet_u$, which contains a total of v tags. The two sets of tags can be represented as two vectors: $\boldsymbol{TagVec}_m = (wg_{m,1}, wg_{m,2}, \ldots, wg_{m,v})$ and $\boldsymbol{TagVec}_n = (wg_{n,1}, wg_{n,2}, \ldots, wg_{n,v})$. The weight $wg_{q,i}$ in these two vectors denotes whether the i-th word appears in question q's tags, which is computed as follows:

$$wg_{q,i} = \frac{y_{q,i}}{\sum_v n_{q,v}}. \tag{4}$$

In (4), $y_{q,i}$, which is either 0 or 1, represents whether the i-th word of $TagSet_u$ appears in the tags of question q or not, $\sum_v n_{q,v}$ denotes the total number of tags in question q, where v is the index of a tag in $TagSet_u$. We measure the similarity between two questions' tags by computing the cosine similarity[10] of their representative vectors $\boldsymbol{TagVec}_m$ and $\boldsymbol{TagVec}_n$. The cosine similarity of two tag vectors is computed in the same way as the cosine similarity of two vectors described in Subsection 3.2.

For a new question nq and an old question oq, we denote their topic similarity score as $TagSim_{nq}(oq)$.

3.6 Composer Component

Consider a new question nq and an old question oq. Their title similarity score, description similarity score, topic similarity score, and tag similarity score are denoted as $TitleSim_{nq}(oq)$, $DesSim_{nq}(oq)$, $TopicSim_{nq}(oq)$, and $TagSim_{nq}(oq)$, respectively. From these four scores, we can compute the composer score, denoted as $Composer_{nq}(oq)$, as follows:

$$\begin{aligned} Composer_{nq}(oq) = {} & \alpha \times TitleSim_{nq}(oq) + \\ & \beta \times DesSim_{nq}(oq) + \\ & \gamma \times TopicSim_{nq}(oq) + \\ & \delta \times TagSim_{nq}(oq). \end{aligned} \tag{5}$$

In (5), $\alpha, \beta, \gamma, \delta \in [0, 1]$ represent the different contribution weights of title similarity, description similarity, topic similarity, and tag similarity to the overall DupPredictor score, respectively. Given a new question nq, nq is compared to the historical questions posted in Stack Overflow earlier, and a set of composer scores are computed. By ranking these historical questions according to their composer scores, we are able to recommend the top-K questions to the new question nq.

To automatically produce good α, β, γ and δ weights for the composer component, we use a sample-based greedy method. Algorithm 1 presents the pseudocode of our sample-based greedy method. Its input includes the set of all questions Q, a training set of

Algorithm 1. EstimateWeights: Estimation of $\alpha, \beta, \gamma, \delta$ in Composer

1: **EstimateWeights**($Q, D, EC, ITER$)
2: **Input:**
3: Q: all question collection
4: D: duplicate question collection
5: EC: evaluation criterion
6: $ITER$: maximum number of iterations (default value = 10)
7: **Output:** $\alpha, \beta, \gamma, \delta$
8: **Method:**
9: **for all** duplicate question $d \in D$, and question $q \in Q$ posted before d **do**
10: Compute their title, description, topic, and tag similarity scores, i.e., $TitleSim_d(q)$, $DesSim_d(q)$, $TopicSim_d(q)$, and $TagSim_d(q)$
11: **end for**
12: **while** iteration count $iter < ITER$ **do**
13: Let $para_1 = 0$, $para_2 = 0$, $para_3 = 0$, $para_4 = 0$;
14: **for all** i from 1 to 4 **do**
15: Choose $para_i = Math.random()$
16: **end for**
17: **for all** i from 1 to 4 **do**
18: $para_i^{\text{best}} = para_i$
19: $para_i = 0$
20: **repeat**
21: Compute the composer scores according to (5)
22: Evaluate the effectiveness of the combined model on D based on EC
23: **if** EC score of $para_i$ is better than that of $para_i^{\text{best}}$ **then**
24: $para_i^{\text{best}} = para_i$
25: **end if**
26: Increase $para_i$ by 0.01
27: **until** $para_i \geqslant 1$
28: $para_i = para_i^{best}$
29: **end for**
30: **end while**
31: Return $para_1, para_2, para_3, para_4$ which give the best result across the $ITER$ iterations based on EC

duplicate questions that were identified before D, an evaluation criterion EC, and the maximum number of iterations $ITER$.

The algorithm first computes the title similarity, description similarity, topic similarity, and tag similarity for each question d in D with questions in Q that are posted before d (lines 9∼11). Next, the algorithm iterates the whole process of choosing good weights $ITER$ times (line 12). In each iteration, the algorithm uses an array *para* to represent the four weights α, β, γ, and δ (line 13). Next, the algorithm randomly assigns a value between 0 to 1 to each $para_i$, for $1 \leqslant i \leqslant 4$ (lines 14∼16). The algorithm then increases $para_i$ incrementally by 0.01 at a time, and computes the EC scores (lines 17∼30). The algorithm would finally return $para_1$, $para_2$, $para_3$, and $para_4$, which represent α, β, γ, and δ respectively, that give the best result based on EC across all $ITER$ iterations (line 31). By default, we set EC as the recall-rate@20, and $ITER$ to 10. Suppose there are m duplicate questions and a total of n historical questions. For each of the duplicate question *dup*, we need to compute the similarity scores for *dup* with each of the n questions. The algorithm complexity in this step is $O(n)$. After we get the similarity scores, we need to rank them, and recommend the top-K questions. The algorithm complexity in this step is $O(n \times \log\ n)$. Thus, the algorithm complexity to get the top-K questions for *dup* is $O(n + n \times \log\ n) = O(n \times \log\ n)$. Since we have m duplicate questions, the total algorithm complexity is $O(m \times n \log\ n)$.

4 Experiments and Results

In this section, we evaluate the effectiveness of DupPredictor. The experimental environment is a Windows Server 2008, 64-bit, Intel® Xeon® 2.00 GHz server with 80 GB of RAM.

4.1 Experimental Setup

We evaluate DupPredictor on historical questions in Stack Overflow. We parse the challenge data published in MSR 2013 mining challenge site[8] [5]. The MSR challenge data contains Stack Overflow data from

[8] http://2013.msrconf.org/challenge.php, July 2015.

2008 to 2012. We extract the first 2 000 000 questions and their corresponding tags. These questions are originally posted between July 2008 and September 2011. We intentionally choose questions that have been published for a long time to ensure that the contents of the questions have stabilized (i.e., no edits are likely to be done any more), the questions are answered and closed, and sufficient time has elapsed to allow more chance for duplicates to be identified by Stack Overflow users and moderators.

We use WVTool[9][13] to extract fields from questions. WVTool is a flexible Java library for statistical language modeling, which is used to create word vector representations of text documents in the vector space model. We use WVTool to remove stop words, do stemming, and produce bags of words from the titles and descriptions of questions. In Stack Overflow, if a question is marked as a duplicate, its title will be appended with the word "Duplicate". The presence of this word will artificially boost the accuracy of DupPredictor. Thus, we delete the word "Duplicate" in the titles of the 1 528 duplicate questions.

In Stack Overflow, users manually detect duplicate questions. Considering the large number of questions posted in Stack Overflow (e.g., more than 2 000 000 questions, and 3 700 new posted questions posted per day[14]), the search space for duplicate questions is extremely large, which makes duplicate question detection a tedious and difficult job. We have searched for questions that are explicitly marked as duplicates in the 2 000 000 questions, and only obtain a set of 1 641 duplicate questions. We manually inspect the 1 641 questions and discover that some questions are incorrectly labeled as duplicates. After removing these wrongly labeled duplicates by checking the questions one by one, we have a total of 1 528 duplicate questions remaining. Given that many developers face similar problems, it is unlikely that only 0.076% of questions in Stack Overflow are duplicates. Rather, it is likely that many duplicates are missed in the manual identification process and remain unidentified in Stack Overflow — we provide an example of such undiscovered duplicate question in Subsection 5.1. The small number of explicitly marked duplicate questions highlights the difficulty of identifying duplicates among the large number of questions in Stack Overflow. To manually identify all duplicate question pairs, four billion comparisons are needed, and if each comparison can be done in five minutes, the process will require 20 billion minutes (many millennia). Thus, there is a need for an automated solution that can help Stack Overflow users and moderators in identifying duplicate questions more effectively and efficiently.

Among the 1 528 duplicate questions, we use the first 300 questions as the training set to train the weights of the composer component of DupPredictor. We use the other 1 228 duplicate questions as the test set to evaluate the effectiveness of DupPredictor. The experimental results indicate that the performance of DupPredictor is relatively stable when the training size is beyond 150 questions. A large number of questions in the training set will cause model overfitting[15], and a long training time in the model training phase. On the other hand, a small number of questions in the training set will cause the model underfitting[15], i.e., the trained model cannot capture the data distribution well. Thus, to balance both the quality of the trained model and the time needed to train a model, we choose 300 duplicates — close to 20% of duplicates in the training set. We pick the first 300 duplicates as the training set to simulate the real usage of our tool — in reality, it is not possible to use future data to predict the past. Table 1 presents the statistics of the training and the test set.

Table 1. Statistics of the Training and the Test Set

Dataset	Period From	Period To	#Duplicate	#All
Training set	2008/7/28	2009/6/12	300	158 164
Test set	2009/6/12	2011/9/07	1 228	1 907 834

Note: #Duplicate refers to the number of duplicate questions. #All refers to the total number of questions.

4.2 Evaluation Metrics

To evaluate the performance of DupPredictor and its four basic components, we use the recall-rate@k metric which was also used in [6, 16-17]. Recall-rate@k is defined as follows:

$$recall\text{-}rate@k = \frac{N_{\text{detected}}}{N_{\text{total}}}. \tag{6}$$

In (6), N_{detected} is the number of duplicate questions whose masters (i.e., original questions that are posted earlier in Stack Overflow) appear in the list of top-K questions, while N_{total} is the total number of duplicate questions used for testing. Recall-rate@k measures the percentage of duplicate questions whose masters are successfully retrieved in the list.

[9] http://sourceforge.net/projects/wvtool/, July 2015.

4.3 Research Questions and Findings

We are interested in answering the following research questions.

*RQ*1 : *How effective is* DupPredictor*? How much improvement could it achieve over existing approaches and its four components?*

Motivation. We need to investigate whether DupPredictor can be used to identify real duplicate questions in Stack Overflow. Also, it is necessary to compare the performance of DupPredictor with the standard search engine of Stack Overflow. Moreover, we need to compare DupPredictor with its four components to show whether the composite method DupPredictor performs better than its constituent components. We also compare DupPredictor with Runeson *et al.*'s approach that has been used to detect duplicate bug reports using natural language processing (NLP) techniques[6]. There are a number of recent studies on duplicate bug report detection, e.g., [16-20]; however, many of them make use of fields in bug reports that are not available in Stack Overflow questions (e.g., Product, Component, Version, and Priority fields), so we choose a classical method to compare with. Answer to this research question shows the benefit of DupPredictor over baseline approaches.

Approach. To answer this research question, we compute recall-rate@5, recall-rate@10, and recall-rate@20 of DupPredictor when it is applied to the Stack Overflow question dataset. We then compare these recall-rates with the recall-rates that can be achieved by using Stack Overflow's search engine, when the title similarity component, description similarity component, topic similarity component, and tag similarity component are used alone, and by Runeson *et al.*'s approach that has been used to detect duplicate bug reports.

To a new question, Stack Overflow recommends 10 questions that appear to be related to the new question, so we can only calculate recall-rate@5 and recall-rate@10 of Stack Overflow's search engine. Fig.3 is an example that presents the 10 related questions of question 2662140, which are recommended by Stack Overflow. We crawl the related question IDs of the 1 228 duplicate questions in the test set to compute the recall-rates of Stack Overflow's search engine. Recall-rate@5 of Stack Overflow's search engine refers to the percentage of questions in the test set whose duplicates are successfully retrieved in the first five related questions recommended by Stack Overflow and recall-rate@10 of Stack Overflow's search engine refers to the percentage of questions in the test set whose duplicates can be found in the 10 related questions recommended by Stack Overflow.

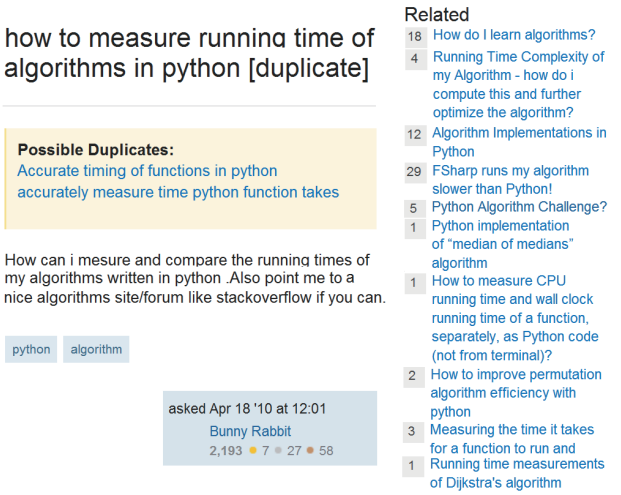


Fig.3. Question 2662140 with its related questions.

Runeson *et al.* detected duplicate defect reports using natural language processing, and they represented defect reports using vector space model[21] and investigated the usage of multiple similarity measures to calculate similarities between defect reports which are represented by their vectors[16]. To use their approach to detect duplicate questions, we combine the title and the description of each question in the test set, and compute its cosine similarity[10] with other questions to find duplicates. According to the experimental result of Runeson *et al.*'s work, cosine similarity performs better than Dice and Jaccard similarity measures[16].

Results. Table 2 presents the recall-rate@5 and recall-rate@10 scores of DupPredictor and the standard search engine of Stack Overflow, and shows the improvements that DupPredictor achieves over Stack Overflow's search engine. From the table, we can see that the recall-rate@5 and the recall-rate@10 of the standard search engine of Stack Overflow are 0.372 and 0.379 respectively. But for our DupPredictor,

Table 2. Recall-Rate@5 and Recall-Rate@10 of DupPredictor Compared with the Standard Search Engine of Stack Overflow (SO), and the Improvements DupPredictor Achieves over the Search Engine

Algorithm	Recall-Rate@5	Recall-Rate@10
DupPredictor	0.423	0.533
SO's search engine	0.372	0.379
Improvement	13.71%	40.63%

the recall-rate@5 and the recall-rate@10 are 0.423 and 0.533 respectively, which outperform Stack Overflow's search engine by 13.71% and 40.63% respectively. Thus, we can draw the conclusion that DupPredictor performs better than Stack Overflow's search engine in terms of recall-rates.

Tables 3∼5 present the recall-rate@5, recall-rate@10, and recall-rate@20 scores of DupPredictor, each of its four components, and Runeson *et al.*'s method, and show the improvements that DupPredictor achieves over the four components and Runeson *et al.*'s method. From Table 3, we find DupPredictor outperforms the four components and Runeson *et al.*'s method in terms of recall-rate@5 by 27.4%, 85.5%, 800%, 264.7%, and 10.2%, respectively. From Table 4, we find that DupPredictor outperforms the four components and Runeson *et al.*'s method in terms of recall-rate@10 by 27.2%, 97.4%, 746%, 231.1%, and 16.4% respectively. From Table 5, we find that DupPredictor outperforms the four components and Runeson *et al.*'s method in terms of recall-rate@20 by 26.1%, 93.9%, 717.9%, 255.5%, and 20.8% respectively.

Table 3. Recall-Rate@5 of DupPredictor Compared with Baselines

Algorithm	Recall-Rate@5	Improvement (%)
DupPredictor	0.423	0.0
Title similarity	0.332	27.4
Description similarity	0.228	85.5
Topic similarity	0.047	800.0
Tag similarity	0.116	264.7
Runeson *et al.*'s method	0.384	10.2

Table 4. Recall-Rate@10 of DupPredictor Compared with Baselines

Algorithm	Recall-Rate@10	Improvement (%)
DupPredictor	0.533	0.0
Title similarity	0.419	27.2
Description similarity	0.270	97.4
Topic similarity	0.063	746.0
Tag similarity	0.161	231.1
Runeson *et al.*'s method	0.458	16.4

Table 5. Recall-Rate@20 of DupPredictor Compared with Baselines

Algorithm	Recall-Rate@20	Improvement (%)
DupPredictor	0.638	0.0
Title similarity	0.506	26.1
Description similarity	0.329	93.9
Topic similarity	0.078	717.9
Tag similarity	0.196	255.5
Runeson *et al.*'s method	0.528	20.8

We can note that DupPredictor can achieve a recall-rate@20 of 63.8% which we believe to be reasonably good. Furthermore, the improvements that DupPredictor achieves over the four components are substantial (26.1%∼800%). These show the effectiveness and benefit of our DupPredictor approach which combines multiple sources of information (i.e., factors) to identify duplicate questions.

*RQ*2 : *What is the effect of varying the number of duplicate questions in the training set on the effectiveness of* DupPredictor*?*

Motivation. DupPredictor takes a training set of questions as input to tune the four parameters of its composer component. By default, this training set includes the first 300 duplicate questions and the other questions that are posted before the 300th duplicate question. In this research question, we perform a sensitivity analysis by investigating the impact of varying the size of the training set on the effectiveness of DupPredictor.

Approach. To answer this research question, we vary the the size of the training set by including the first 100∼500 duplicate questions and the other questions posted before the last of the duplicate questions, and record the recall-rate@5, recall-rate@10, recall-rate@20 scores. Considering that there are 1 528 duplicate questions in our dataset, when we set the number of duplicate questions in the training set to n, we would use the remaining $1\,528 - n$ duplicate questions as the test set.

Results. Fig.4 presents the recall-rate@5, recall-rate@10, and recall-rate@20 scores of DupPredictor for training sets containing different numbers of duplicate questions. We notice that when the number of questions in the training set is too small (i.e., 100 duplicate questions), the effectiveness of DupPredictor is reduced. However, beyond a certain training size threshold (i.e., 150 duplicate questions or more), the performance of DupPredictor is relatively stable. For example, the recall-rate@5 only varies from 0.413 to 0.425 when the training set is varied to include 150∼500 duplicate questions (and other questions posted before the last of the duplicate question). Thus, in practice, we suggest to set the number of duplicate questions in the training set to 150 or more.

Fig.5 presents the α, β, γ and δ weights learned by DupPredictor with varying number of duplicate questions in the training set. We notice for different numbers of duplicate questions in the training set, the learned weights are different. However, there is a trend:

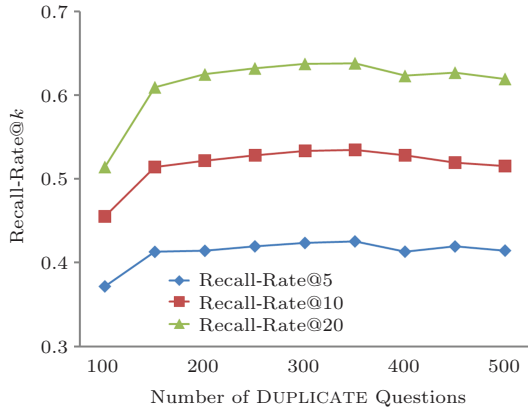


Fig.4. Recall-rate@k of DUPPREDICTOR when the size of the training set is varied.

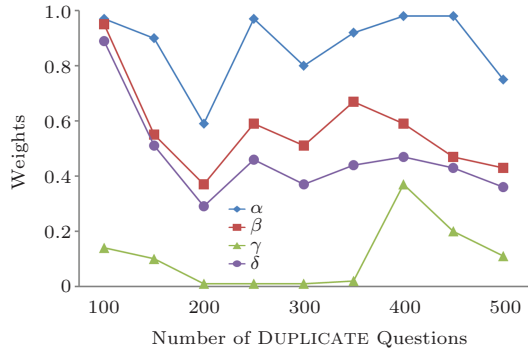


Fig.5. α, β, γ and δ weights of DUPPREDICTOR for different numbers of duplicate questions in the training set.

$\alpha > \beta > \delta > \gamma$. The stable performance of DUPPREDICTOR for many different numbers of duplicate questions (i.e., beyond 150) indicates that it can adapt well to different numbers of duplicate questions in the training set.

RQ3 : Does DUPPREDICTOR *estimate the four weights (i.e., α, β, γ, δ) that govern the relative contribution of its four constituent components well? What is the effect of varying the four weights on the effectiveness of* DUPPREDICTOR*?*

Motivation. By default, DUPPREDICTOR automatically estimates α, β, γ and δ weights. In this research question, we would like to investigate whether DUPPREDICTOR estimates the four weights well. We also would like to analyze the effect of using different sets of weights on the effectiveness of DUPPREDICTOR.

Approach. In order to analyze the effect of the settings of the four weights to the effectiveness of DUPPREDICTOR, we randomly generate 50 sets of weights and use them to determine the contributions of the four components of DUPPREDICTOR. We compare the effectiveness of using these 50 sets of weights with the set of weights that is automatically estimated by DUPPREDICTOR in terms of recall-rate@5, recall-rate@10, and recall-rate@20.

Results. Table 6 presents the recall-rate@5, recall-rate@10, and recall-rate@20 of the set of weights learned by DUPPREDICTOR in its default setting (i.e., the first row), and the 50 sets of weights generated randomly. DUPPREDICTOR estimates the four weights to be: $\alpha = 0.8$, $\beta = 0.51$, $\gamma = 0.01$, and $\delta = 0.37$. The recall-rate@5, recall-rate@10, and recall-rate@20 of this set of weights are 0.423, 0.533, and 0.638, respectively, which are the best results in the table. This shows that DUPPREDICTOR can estimate a good set of weights. Among the 50 sets of weights, some sets yield poor results (recall-rate@20 < 0.4), e.g., $\alpha = 0.45$, $\beta = 0.07$, $\gamma = 0.74$ and $\delta = 0.28$, while some yield reasonably good results (recall-rate@20 > 0.5), e.g., $\alpha = 0.65$, $\beta = 0.41$, $\gamma = 0.24$, and $\delta = 0.39$. Suitable tuning of the weights is important to the effectiveness of DUPPREDICTOR.

From Table 6, we can also find that in general when $\alpha > \beta > \delta > \gamma$, recall-rate@5, recall-rate@10 and recall-rate@20 are better. This indicates that for identifying duplicate questions, title similarity is more important than description similarity, which in turn is more important than tag similarity. Furthermore, topic similarity alone cannot be used to identify duplicate questions well.

5 Discussion

5.1 Undiscovered Duplicate Question

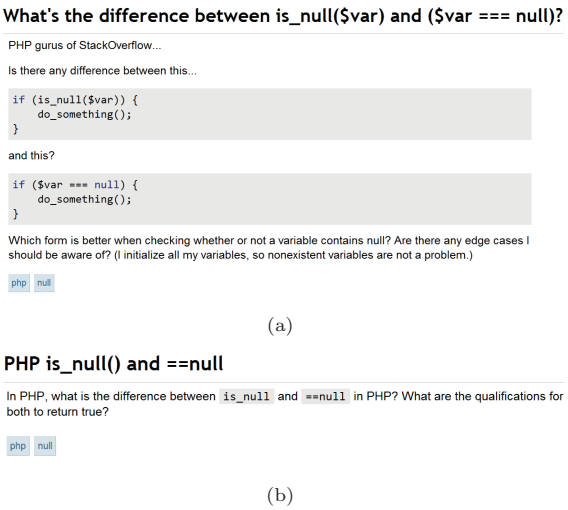
In our dataset, among the more than two million questions, we only find 1 528 duplicate questions. To understand why this is the case, we manually investigate many Stack Overflow questions. We find that many duplicate questions have not been discovered and marked as such by users. Fig.6 is an example of two duplicate questions that have not been identified yet. Both questions are asking about the difference between "*is_null*()" and "== *null*" in PHP. This shows the need for DUPPREDICTOR. By using DUPPREDICTOR, many duplicate questions, like question 4662588[10] and question 9671659[11], could be identified early.

[10] http://stackoverflow.com/questions/4662588, July 2015.
[11] http://stackoverflow.com/questions/9671659, July 2015.

Table 6. Recall-Rate@5, Recall-Rate@10 and Recall-Rate@20 of DupPredictor with Different Weights Assigned to the Four Basic Components

α	β	γ	δ	R@5	R@10	R@20
0.80	**0.51**	**0.01**	**0.37**	**0.423**	**0.533**	**0.638**
0.41	0.24	0.61	0.92	0.258	0.302	0.327
0.31	0.34	0.11	0.35	0.361	0.430	0.473
0.45	0.07	0.74	0.28	0.130	0.143	0.152
0.10	0.51	0.81	0.17	0.113	0.121	0.126
0.94	0.88	0.95	0.77	0.270	0.340	0.432
0.66	0.31	0.28	0.45	0.406	0.517	0.612
0.64	0.70	0.35	0.10	0.318	0.419	0.512
0.50	0.27	0.22	0.34	0.406	0.519	0.606
0.75	0.42	0.57	0.24	0.305	0.406	0.501
0.99	0.91	0.19	0.29	0.407	0.533	0.623
0.98	0.70	0.84	0.26	0.258	0.345	0.445
0.74	0.04	0.26	0.45	0.384	0.475	0.562
0.30	0.94	0.12	0.67	0.327	0.405	0.456
0.18	0.46	0.41	0.96	0.222	0.261	0.286
0.51	0.71	0.19	0.97	0.274	0.322	0.350
0.40	0.43	0.07	0.62	0.282	0.335	0.361
0.46	0.63	0.77	0.19	0.147	0.182	0.231
0.32	0.12	0.49	0.91	0.219	0.264	0.288
0.50	0.50	0.15	0.44	0.393	0.476	0.535
0.82	0.73	0.36	0.81	0.372	0.449	0.505
0.51	0.97	0.01	0.84	0.352	0.415	0.455
0.43	0.17	0.74	0.97	0.256	0.300	0.327
0.65	0.41	0.24	0.39	0.409	0.529	0.622
0.10	0.94	0.91	0.97	0.184	0.215	0.265
0.62	0.45	0.06	0.61	0.361	0.434	0.482
0.81	0.95	0.76	0.54	0.263	0.355	0.456
0.08	0.48	0.70	0.90	0.216	0.260	0.298
0.94	0.69	0.99	0.04	0.178	0.222	0.288
0.49	0.22	0.43	0.14	0.257	0.328	0.419
0.63	0.81	0.18	0.47	0.395	0.490	0.552
0.64	0.80	0.77	0.95	0.277	0.331	0.386
0.58	0.97	0.44	0.88	0.324	0.395	0.442
0.48	0.31	0.08	0.41	0.390	0.480	0.550
0.15	0.74	0.24	0.05	0.192	0.272	0.360
0.08	0.57	0.90	0.40	0.116	0.130	0.140
0.50	0.69	0.76	0.54	0.206	0.262	0.354
0.87	0.67	0.74	0.89	0.324	0.387	0.447
0.20	0.06	0.57	0.73	0.225	0.272	0.301
0.45	0.42	0.25	0.58	0.313	0.371	0.410
0.70	0.64	0.75	0.43	0.233	0.305	0.393
0.61	0.83	0.81	0.89	0.257	0.318	0.382
0.31	0.72	0.69	0.94	0.241	0.282	0.325
0.38	0.39	0.15	0.28	0.392	0.489	0.565
0.86	0.25	0.08	0.96	0.274	0.311	0.346
0.21	0.45	0.44	0.25	0.170	0.217	0.281
0.02	0.65	0.88	0.56	0.123	0.150	0.177
0.68	0.24	0.94	0.50	0.191	0.221	0.244
0.96	0.72	0.41	0.05	0.357	0.467	0.555
0.19	0.33	0.24	0.24	0.247	0.322	0.420
0.94	0.80	0.66	0.64	0.340	0.451	0.534

Note: R means recall-rate.



(a) (b)

Fig.6. Example of duplicate questions that have not been identified in Stack Overflow. (a) Question 4662588. (b) Question 9671659.

5.2 Time Efficiency

The efficiency of the algorithm will affect its practical usage. Thus, in this research question, we investigate the time efficiency of DupPredictor. We run DupPredictor and report the average model training and test time. Model training time refers to the time taken to build the composer component. Test time refers to the time taken for DupPredictor to predict the duplicates of the questions in the test set. We notice that the training time and the test time of DupPredictor are reasonable, e.g., on average, we need about 725.542 seconds to train a model, and 10.858 seconds to predict the duplicates of all questions in the test set using the model (for each question in the test set, we just need 0.008 8 seconds). Notice that the training phase can be done offline (e.g., overnight) and the model does not need to be updated all the time. A trained model can be used to predict many questions.

5.3 Impact of Topic Similarity Component

From the training results, we notice that the weight of topic similarity component is only 0.01, much smaller than those of the other components. Therefore, we need to investigate whether the topic similarity component makes sense in DupPredictor. We build a model without topic similarity component, called DupPredictor$^{-\text{TOPIC}}$, which only combines title similarity component, description similarity component, and tag similarity component. Then, we run DupPredictor$^{-\text{TOPIC}}$ on the same dataset and compute recall-rate@5, recall-rate@10 and recall-rate@20.

The results are shown in Table 7. From the table, we can see that without topic similarity component, the recall-rate@5, recall-rate@10 and recall-rate@20 scores are slightly reduced. Since topic similarity component could still help to improve the performance of DupPredictor, in practice, we recommend users to include this component.

Table 7. Recall-Rate@5, Recall-Rate@10 and Recall-Rate@20 of DupPredictor$^{-\text{TOPIC}}$ Compared with DupPredictor

Algorithm	α	β	γ	δ	R@5	R@10	R@20
DupPredictor	0.80	0.51	0.01	0.37	0.423	0.533	0.638
DupPredictor$^{-\text{TOPIC}}$	0.87	0.59	0.00	0.29	0.414	0.528	0.631

Note: R means recall-rate.

5.4 Threats to Validity

There are several threats that may potentially affect the validity of our study. Threats to internal validity relate to errors in our experiments and implementation. We have double checked our experiments and implementation. We have also manually checked the questions marked as duplicates in our dataset to ensure that they are really duplicates. Still, there could be errors that we have not noticed. Threats to external validity relate to the generalizability of our results. We have analyzed 1 528 duplicate questions from more than 2 000 000 questions in Stack Overflow. In the future, we plan to reduce this threat further by analyzing even more duplicate questions from additional question and answer sites. Threats to construct validity refer to the suitability of our evaluation metrics. We use recall-rate@5, recall-rate@10, and recall-rate@20 as the evaluation metrics. These metrics are also used by many previous studies[6,16-17]. Thus, we believe there is little threat to construct validity.

6 Related Work

In this section, we briefly review related studies. We first review some previous studies on duplicate bug report detection. Next, we describe studies on software information sites and online forum discussion.

Studies on Duplicate Bug Report Detection. There have been a number of studies on duplicate bug report detection, e.g., [6, 16-20]. The motivation of these studies is to reduce the workload of bug triagers who need to identify duplicate bug reports among the hundreds of bug reports that they receive daily[23]. Most of the duplicate bug report detection approaches take a new bug report as input, and recommend a list of top-k existing bug reports that are the most similar to it. A bug triager can inspect this top-k list and make an informed decision whether the bug report is a duplicate one or not. We highlight some of the more recent studies on duplicate bug report detection below.

Wang *et al.* identified duplicate defect reports using execution trace information (i.e., list of executed methods) of bug-revealing runs and natural language information contained in bug reports[18]. Sun *et al.* proposed a discriminative model based approach for duplicate bug report detection[17]. They extracted a number of features from duplicate and non-duplicate bug reports and processed these features using Support Vector Machine (SVM) to create a discriminative model that can compute the likelihood of a bug report to be a duplicate of another report. These likelihood scores are then used to rank existing bug reports given a new bug report. In their other work, Sun *et al.* proposed a retrieval function, named REP, to measure the similarity between two bug reports[16]. REP extends BM25F, which is an effective retrieval function proposed in the information retrieval (IR) community, by considering a special characteristic of duplicate bug reports.

Aside from the above-mentioned studies, there are also a number of other studies that focus on the task of predicting if a pair of bug reports are duplicate of each other or not. These include studies by Lo *et al.*[24], Alipour *et al.*[19], Lazar *et al.*[22], and Klein *et al.*[20]

Different from the above mentioned studies, in this work, we address a different problem, namely the detection of duplicate questions in Stack Overflow. There are a number of differences between duplicate bug report detection and duplicate question detection.

1) To identify duplicate bug reports, there is no need to look too far behind; the duplicate of a new bug report is likely to be reported a short period of time before the new bug report. This is the case since after some time, a bug report will be fixed, and thus the bug will no longer be experienced by users. Hence, after some time period has elapsed, it is likely that no future duplicate bug report will be made. Many duplicate bug report detection approaches make use of this special characteristic of bug reports, e.g., [16, 18]. Different from duplicate bug reports, duplicate questions in Stack Overflow can be separated by a long time interval. Fig.7 is an example of two duplicate questions that are separated by a long time interval. Question 7328545[12] was asked on September 7, 2011 and was labeled as a duplicate ques-

tion of question 1150321[13], which was asked on July 19, 2009. The time interval between the two reports is more than two years.

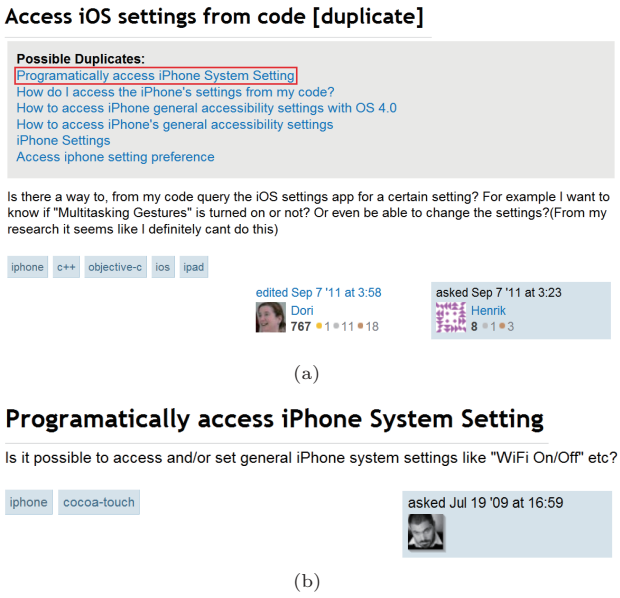


Fig.7. Example of duplicate questions that are separated by a long time interval in Stack Overflow. (a) Question 7328545. (b) Question 1150321.

2) Bug reports and questions in Stack Overflow contain different fields. A bug report contains fields such as severity, priority, component, product, and so on. Past duplicate bug report detection approaches make use of this special characteristic of bug reports, e.g., [16]. Different from bug reports, Stack Overflow questions do not have many of the fields that appear in bug reports. Furthermore, these questions have tags that do not appear in bug reports. DupPredictor makes use of these tags to help identify duplicate questions.

Due to the above mentioned differences, there is a need for a new approach to detecting duplicate questions in Stack Overflow.

Studies on Bug Report Management. Zanetti *et al.* proposed an efficient and practical method to identify valid bug reports, which is based on nine measures to quantify the social embeddedness of bug reporters in the collaboration network[25]. Xuan *et al.* combined instance selection with feature selection to simultaneously reduce data scale on the bug dimension and the word dimension, and the experimental results show that their data reduction can effectively reduce the data scale and improve the accuracy of bug triage[26].

Studies on Software Information Sites. There have been a number of studies on software information sites. We use the term software information sites loosely to refer to web-based channels that developers have used to share information and help one another. These include question and answer sites, forums, microblogging sites, etc. We highlight some of these studies below.

Storey *et al.*[3] and Begel *et al.*[2] wrote two position papers about the outlook of research in social media for software engineering. They proposed a set of research questions related to the impact of social media for software engineering at team, project, and community levels. Bougie *et al.*[27] and Tian *et al.*[28] analyzed microblogs related to software engineering activities to understand what software engineers do in Twitter. Prasetyo *et al.* proposed an approach that can automatically classify a microblog as either software-related or not[29].

Surian *et al.* found collaboration patterns in SourceForge.Net by employing a novel combination of graph mining and graph matching techniques[30]. Surian *et al.* used random walk with restart (RWR) method to build a large-scale developer collaboration network to recommend suitable developers by using information collected from SourceForge.Net[31]. Hong *et al.* compared developer social networks and popular general social networks and examined how developer social networks evolve over time[32].

Xia *et al.* proposed an automatic tag recommendation method which analyzes objects in Stack Overflow and Freecode and recommends tags to them[1]. Wang *et al.* extended the work of Xia *et al.*[1] by proposing an approach that combines frequentists and Bayesian inference to better recommend tags to objects in StackExchange websites[33]. Barua *et al.* used LDA to automatically discover the main topics of questions in Stack Overflow[34]. Gottipati *et al.* provided a semantic search engine to infer semantic tags of posts and recover relevant answer posts in software forums[35]. Henβ *et al.* presented an approach that automatically extracts frequently asked questions from software development discussions by combining techniques of text mining and natural language processing[36]. Correa and Sureka used a machine learning framework to build a predictive model to judge whether a Stack Overflow question would be closed or not[37]. They also built a predictive model to detect whether a question will be deleted or not[14]. We also found that many forums

[12] http://stackoverflow.com/questions/7328545, July 2015.
[13] http://stackoverflow.com/questions/1150321, July 2015.

provide a list of related questions to a question, such as IBM's jazz forum[14] and Stack Overflow, and we compare our DupPredictor with the method used to detect related questions in Stack Overflow in our experiment.

Our work is orthogonal to the above mentioned studies: we focus on identifying duplicate questions in Stack Overflow, which is different from the focus of the above mentioned studies. Among the previous studies, the one closest to our work is Correa and Sureka's studies[13,37], which predict if a question will be closed or deleted. However, duplicate questions are different from deleted questions, and duplicate questions are only a small part of closed questions, about 29.3% according to Correa and Sureka's work[37]. What is more, different from Correa and Sureka's studies, our work not only can be used to determine if a current question is a duplicate question, but also can find the questions of which the current question is a duplicate. It would be hard for a developer to trust the output of a black box machine learning prediction tool (especially since the accuracy of such tool is not 100%). To deal with this issue, our approach gives evidences that developers can consider to decide if a question is a duplicate one, i.e., by providing a list of existing questions that are likely to be duplicates of the target question.

Online Forum Discussion. In Stack Overflow, a large quantity of questions are related to APIs. There have been several studies on API discussions in online forums. Zhou *et al.* proposed a cache-based composite algorithm to automatically categorize API discussions[38]. Hou and Mo applied machine learning algorithms to categorize API discussions based on their contents in software forums, and they concluded that multinomial naive Bayes achieves a remarkably high accuracy[39]. Hou and Li studied 172 API discussions in Swing forums to analyze the root causes of API usage obstacles and discussed what could be done to help overcome these obstacles[40]. Rupakheti and Hou built a critic to advise the usage of an API, and performed an empirical study on a Java Swing forum to assess to what extent the critic can help solve practical API usage problems[41]. Zhang and Hou investigated ways to extract problematic API features from online forums, using natural language processing and sentiment analysis techniques[42]. Our work also focuses on problems in online forums, and aims at optimizing the user experience.

7 Conclusions

In this paper, we proposed DupPredictor to identify duplicate questions in Stack Overflow. DupPredictor measures the similarity of two questions by comparing their observable factors, which are the titles, descriptions, and tags of the questions, and their latent factors corresponding to the topic distributions that are learned from the natural language descriptions of the questions. DupPredictor has four components: title similarity component, description similarity component, topic similarity component, and tag similarity component. It automatically combines these four components by assigning different weights to them; these weights are automatically learned from a training data. We evaluated DupPredictor on more than two million questions in Stack Overflow site and measured the performance of DupPredictor in terms of recall-rate@k. The experimental results show that DupPredictor can achieve recall-rate@5, recall-rate@10, and recall-rate@20 scores of 42.3%, 53.3%, and 63.8%, respectively. We compare DupPredictor with its four constituent components, the standard search engine of Stack Overflow, and Runeson *et al.*'s approach that has been used to detect duplicate bug reports, and the results showed that DupPredictor improves these baseline approaches by 10.2%∼717.9%.

In the future, we plan to evaluate DupPredictor with datasets from other software question and answer sites or forums, and develop a better technique to improve the effectiveness of DupPredictor.

References

[1] Xia X, Lo D, Wang X, Zhou B. Tag recommendation in software information sites. In *Proc. the 10th Working Conference on Mining Software Repositories (MSR)*, May 2013, pp.287-296.

[2] Begel A, DeLine R, Zimmermann T. Social media for software engineering. In *Proc. the FSE/SDP Workshop on Future of Software Engineering Research*, November 2010, pp.33-38.

[3] Storey M A, Treude C, Deursen A, Cheng L T. The impact of social media on software engineering practices and tools. In *Proc. the FSE/SDP Workshop on Future of Software Engineering Research*, November 2010, pp.359-364.

[4] Blei D M, Ng A Y, Jordan M I. Latent Dirichlet allocation. *Journal Machine Learning Research*, 2003, 3: 993-1022.

[5] Bacchelli A. Mining challenge 2013: Stack Overflow. In *Proc. the 10th MSR*, May 2013.

[6] Runeson P, Alexandersson M, Nyholm O. Detection of duplicate defect reports using natural language processing. In

[14] https://jazz.net/forum, July 2015.

Proc. the 29th International Conference on Software Engineering (*ICSE*), May 2007, pp.499-510.

[7] Porter M. An algorithm for suffix stripping. *Program*, 1980, 14(3): 130-137.

[8] Kochhar P S, Thung F, Lo D. Automatic fine-grained issue report reclassification. In *Proc. the 19th International Conference on Engineering of Complex Computer Systems* (*ICECCS*), August 2014, pp.126-135.

[9] Thung F, Lo D, Jiang L. Automatic defect categorization. In *Proc. the 19th Working Conference on Reverse Engineering* (*WCRE*), October 2012, pp.205-214.

[10] Baeza-Yates R, Ribeiro-Neto B. Modern Information Retrieval: The Concepts and Technology Behind Search (2nd edition). Addision Wesley, 2011.

[11] Heinrich G. Parameter estimation for text analysis. Technical Report, University of Leipzig, 2005. http://www.arbulon.net/publications/text-est.pdf, Aug. 2015.

[12] Steyvers M, Griffiths T. Probabilistic topic models. In *Handbook of Latent Semantic Analysis*, Landauer T, Mcnamara D, Dennis S *et al.* (eds.), Routledge, 2007.

[13] Wurst M. The word vector tool user guide operator reference developer tutorial. http://www-ai.cs.uni-dortmund.de/SOFTWARE/WVTOOL/doc/wvtool-1.0.pdf, July 2015.

[14] Correa D, Sureka A. Chaff from the wheat: Characterization and modeling of deleted questions on Stack Overflow. In *Proc. the 23rd International Conference on World Wide Web*, April 2014, pp.631-642.

[15] Han J, Kamber M. Data Mining: Concepts and Techniques (2nd edition). San Francisco, CA, USA: Morgan Kaufmann, 2006.

[16] Sun C, Lo D, Khoo S C, Jiang J. Towards more accurate retrieval of duplicate bug reports. In *Proc. the 26th IEEE/ACM International Conference on Automated Software Engineering*, November 2011, pp.253-262.

[17] Sun C, Lo D, Wang X, Jiang J, Khoo S C. A discriminative model approach for accurate duplicate bug report retrieval. In *Proc. the 32nd ICST, Volume 1*, May 2010, pp.45-54.

[18] Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. In *Proc. the 30th International Conference on Software Engineering*, May 2008, pp.461-470.

[19] Alipour A, Hindle A, Stroulia E. A contextual approach towards more accurate duplicate bug report detection. In *Proc. the 10th MSR*, May 2013, pp.183-192.

[20] Klein N, Corley C S, Kraft N A. New features for duplicate bug detection. In *Proc. the 11th MSR*, May 31-June 1, 2014, pp.324-327.

[21] Manning C D, Raghavan P, Schütze H. Introduction to Information Retrieval, Volume 1. Cambridge University Press Cambridge, 2008.

[22] Lazar A, Ritchey S, Sharif B. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proc. the 11th MSR*, May 31-June 1, 2014, pp.308-311.

[23] Anvik J, Hiew L, Murphy G C. Coping with an open bug repository. In *Proc. the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, October 2005, pp.35-39.

[24] Lo D, Cheng H, Lucia. Mining closed discriminative dyadic sequential patterns. In *Proc. the 14th International Conference on Extending Database Technology* (*EDBT*), March 2011, pp.21-32.

[25] Zanetti M S, Scholtes I, Tessone C J, Schweitzer F. Categorizing bugs with social networks: A case study on four open source software communities. In *Proc. the 35th ICSE*, May 2013, pp.1032-1041.

[26] Xuan J, Jiang H, Hu Y, Ren Z, Zou W, Luo Z, Wu X. Towards effective bug triage with software data reduction techniques. *IEEE Transactions on Knowledge and Data Engineering*, 2015, 27(1): 264-280.

[27] Bougie G, Starke J, Storey M A, German D M. Towards understanding Twitter use in software engineering: Preliminary findings, ongoing challenges and future questions. In *Proc. the 2nd International Workshop on Web 2.0 for Software Engineering*, May 2011, pp.31-36.

[28] Tian Y, Achananuparp P, Lubis I N, Lo D, Lim E P. What does software engineering community microblog about? In *Proc. the 9th MSR*, June 2012, pp.247-250.

[29] Prasetyo P K, Lo D, Achananuparp P, Tian Y, Lim E P. Automatic classification of software related microblogs. In *Proc. the 28th ICSM*, September 2012, pp.596-599.

[30] Surian D, Lo D, Lim E P. Mining collaboration patterns from a large developer network. In *Proc. the 17th Working Conference on Reverse Engineering* (*WCRE*), October 2010, pp.269-273.

[31] Surian D, Liu N, Lo D, Tong H, Lim E P, Faloutsos C. Recommending people in developers' collaboration network. In *Proc. the 18th WCRE*, October 2011, pp.379-388.

[32] Hong Q, Kim S, Cheung S, Bird C. Understanding a developer social network and its evolution. In *Proc. the 27th IEEE International Conference on Software Maintenance* (*ICSM*), September 2011, pp.323-332.

[33] Wang S, Lo D, Vasilescu B, Serebrenik A. EnTagRec: An enhanced tag recommendation system for software information sites. In *Proc. the 30th ICSME*, September 29-October 31 2014, pp.291-300.

[34] Barua A, Thomas S W, Hassan A E. What are developers talking about? An analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 2014, 19(3): 619-654.

[35] Gottipati S, Lo D, Jiang J. Finding relevant answers in software forums. In *Proc. the 26th IEEE/ACM International Conference on Automated Software Engineering*, November 2011, pp.323-332.

[36] Henβ S, Monperrus M, Mezini M. Semi-automatically extracting FAQs to improve accessibility of software development knowledge. In *Proc. the 34th ICSE*, June 2012, pp.793-803.

[37] Correa D, Sureka A. Fit or unfit: Analysis and prediction of 'closed questions' on stack overflow. In *Proc. the 1st ACM Conference on Online Social Networks*, October 2013, pp.201-212.

[38] Zhou B, Xia X, Lo D, Tian C, Wang X. Towards more accurate content categorization of API discussions. In *Proc. the 22nd International Conference on Program Comprehension*, June 2014, pp.95-105.

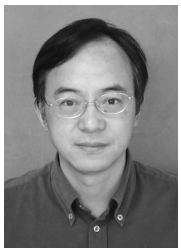
[39] Hou D, Mo L. Content categorization of API discussions. In *Proc. the 29th ICSM*, September 2013, pp.60-69.

[40] Hou D, Li L. Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. In *Proc. the 19th IEEE International Conference on Program Comprehension* (*ICPC*), June 2011, pp.91-100.
[41] Rupakheti C R, Hou D. Evaluating forum discussions to inform the design of an API critic. In *Proc. the 20th ICPC*, July 2012, pp.53-62.
[42] Zhang Y, Hou D. Extracting problematic API features from forum discussions. In *Proc. the 21st ICPC*, May 2013, pp.142-151.

Yun Zhang is a Ph.D. candidate in the College of Computer Science and Technology, Zhejiang University, Hangzhou. Her research interests include mining software repository and empirical study.

David Lo received his Ph.D. degree in computer science from the School of Computing, National University of Singapore, Singapore, in 2008. He is currently an assistant professor in the School of Information Systems, Singapore Management University. He has close to 10 years of experience in software engineering and data mining research and has more than 130 publications in these areas. He received the Lee Foundation Fellow for Research Excellence from the Singapore Management University in 2009. He has won a number of research awards including an ACM Distinguished Paper Award for his work on bug report management. He has published in many top international conferences in software engineering, programming languages, data mining and databases, including ICSE, FSE, ASE, PLDI, KDD, WSDM, TKDE, ICDE, and VLDB. He has also served on the program committees of ICSE, ASE, KDD, VLDB, and many others. He is a steering committee member of the IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER) which is a merger of the two major conferences in software engineering, namely CSMR and WCRE. He will also serve as the general chair of ASE 2016. He is a leading researcher in the emerging field of software analytics and has been invited to give keynote speeches and lectures on the topic in many venues, such as the 2010 Workshop on Mining Unstructured Data, the 2013 Génie Logiciel Empirique Workshop, the 2014 International Summer School on Leading Edge Software Engineering, and the 2014 Estonian Summer School in Computer and Systems Science.

Xin Xia received his Ph.D. degree in computer science from the College of Computer Science and Technology, Zhejiang University, Hangzhou, in 2014. He is currently a research assistant professor in the College of Computer Science and Technology at Zhejiang University. His research interests include software analytic, empirical study, and mining software repository.



Jian-Ling Sun received his Ph.D. degree in computer science from Zhejiang University, Hangzhou, in 1993. He is currently a professor in the College of Computer Science, Zhejiang University. His research interests include database systems, distributed systems, and software engineering.