Duplicate Question Detection in Stack Overflow: A Reproducibility Study

Rodrigo F. G. Silva
Federal University of Uberlândia
Uberlândia (MG), Brazil
rodrigofernandes@ufu.br

Klérisson Paixão
Federal University of Uberlândia
Uberlândia (MG), Brazil
klerisson@ufu.br

Marcelo de A. Maia
Federal University of Uberlândia
Uberlândia (MG), Brazil
marcelo.maia@ufu.br

***Abstract*—Stack Overflow has become a fundamental element of developer toolset. Such influence increase has been accompanied by an effort from Stack Overflow community to keep the quality of its content. One of the problems which jeopardizes that quality is the continuous growth of duplicated questions. To solve this problem, prior works focused on automatically detecting duplicated questions. Two important solutions are *DupPredictor* and *Dupe*. Despite reporting significant results, both works do not provide their implementations publicly available, hindering subsequent works in scientific literature which rely on them. We executed an empirical study as a reproduction of *DupPredictor* and *Dupe*. Our results, not robust when attempted with different set of tools and data sets, show that the barriers to reproduce these approaches are high. Furthermore, when applied to more recent data, we observe a performance decay of our both reproductions in terms of recall-rate over time, as the number of questions increases. Our findings suggest that the subsequent works concerning detection of duplicated questions in Question and Answer communities require more investigation to assert their findings.**

***Index Terms*—Stack Overflow; Question quality; Duplicate questions; Classification;**

I. Introduction

Stack Overflow is a fundamental element of the toolset of programmers nowadays [1]–[4]. It offers a broad knowledge base for software developers and is one of the dominant domain-specific Question and Answer (Q&A) platforms on the Web [5], [6]. Such success is fostered by "the wisdom of crowds" [7], where millions of people ask and answer questions to produce and share relevant programming related content [8]–[12].

However, prior research suggests a quality decay in the platform [13], especially because of the rampant growth in the number of duplicated questions [14]–[18]. One goal of preventing duplicates is to help people find right answers in one place. Duplicates pollute the platform with questions already answered.

In practice, Stack Overflow uses numerous mechanisms to prevent duplicates.[1,2,3] Because users are energized by the wish to learn or to burnish one's reputation [19], much effort focus on gamification incentives driving user participation to eliminate duplicates [20]. Besides incentives, there exist automated solutions to detect duplicates. But, we keep seeing excessive duplicates cluttering up over time [15]. For instance, Figure 1 shows two questions from Stack Overflow, where question 1(a) was appointed by the community as the duplicate of question 1(b). These questions are not copy and paste, and even observing that their textual features are completely different, we realize that they cover the exact same ground.

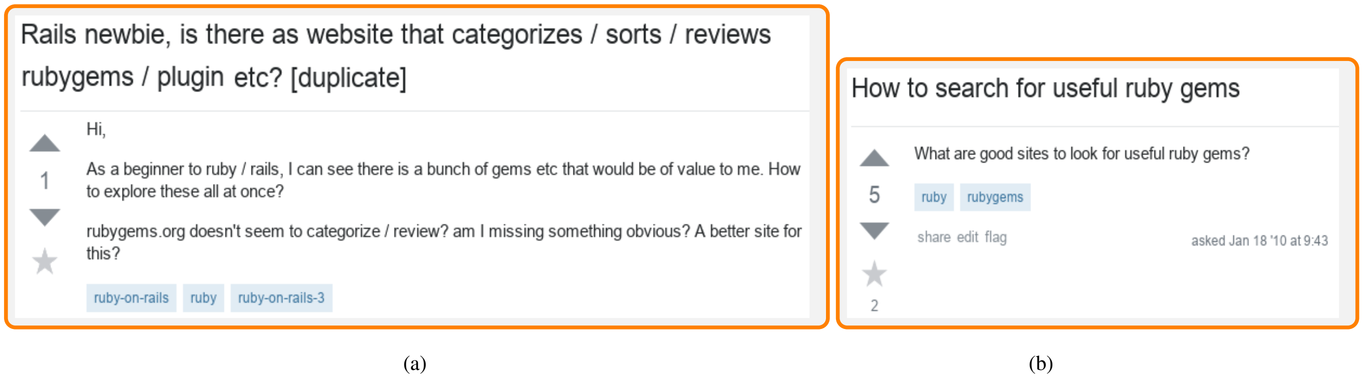
Looking at academic research, few works have sought to propose duplicate question detection for Stack Overflow. In particular, Zhang et al. [14] envisioned *DupPredictor*, an approach to predict whether a question is a duplicate; Then, Ahasanuzzaman et al. [15] propose *Dupe*, which also extracted features from the question corpus to build a question pair binary classifier (duplicated or not).

We designed this study as an independent reproduction of *DupPredictor* and *Dupe*, here called *DupPredictorRep* and *DupeRep* respectively, meaning that we developed the study artifacts independently. Hence, there is a possible implementation bias, as it reflects our interpretation of the prior works. Whereas neither approaches are publicly available, we provide our implementations[4,5] to community along with the data set we used[6] to enable other researchers to repeat, improve, or refute our results.

Our main goal is not only to confirm or reject previous findings, but to extend the results by considering newer data sets. This reproduction differs from the original works because the artifacts are re-implemented and different data sets are used. *DupPredictor* was evaluated using the MSR'13 Challenge track data set [21], whereas *Dupe* used the MSR'15 one [22]. Instead of reproducing each approach with their respective dataset, we decided to use only one dataset to have the same data to compare both reproductions, and also to extend the results to more recent years. So, we rely on the newest dump (March, 2017), and filter questions by creation data for artificially simulating data sets for different years.

Our findings suggest that the barriers for reproduction of duplicate detection are still high. First, because implementations are not available, a concern already raised by the Mining Software Repository research community [23], [24]. Second, because our results differ from the original works in multiple

[1]http://stackoverflow.blog/2008/12/31/i-move-to-close-this-question/
[2]https://stackoverflow.blog/2009/04/29/handling-duplicate-questions/
[3]https://stackoverflow.blog/2010/06/10/improved-question-merging/
[4]https://github.com/muldon/dupPredictorRep
[5]https://github.com/muldon/dupeRep
[6]https://goo.gl/ATJgYp



Rails newbie, is there as website that categorizes / sorts / reviews rubygems / plugin etc? [duplicate]

1

Hi,

As a beginner to ruby / rails, I can see there is a bunch of gems etc that would be of value to me. How to explore these all at once?

rubygems.org doesn't seem to categorize / review? am I missing something obvious? A better site for this?

ruby-on-rails ruby ruby-on-rails-3

(a)

How to search for useful ruby gems

5

What are good sites to look for useful ruby gems?

ruby rubygems

share edit flag

asked Jan 18 '10 at 9:43

2

(b)

Fig. 1. Stack Overflow is facing a growing number of duplicated questions. Here, question (a) was appointed as duplicate of question (b). However, their textual features are completely different, challenging automated solutions.

aspects. For the same scenario proposed by the original works, *DupPredictorRep* and *DupeRep* show a decrease for recall-rate, in absolute values, up to 20.5% and 28.5%, respectively. Notwithstanding, our results show that *DupPredictorRep* surpass *DupeRep* at recall-rate@20 (Table V), contradicting *Dupe*'s original results. Over time, however, *DupeRep* obtains more stable predictions, whereas *DupPredictorRep* shows a severe decay.

Hence, our contributions include:

- Reproducible experiments of *DupPredictor* and *Dupe* on duplicate question detection with available data and artifacts, showing lower recall-rates compared to those reported in the original works.
- A comparison of both approaches over the time, considering the growth of number of questions.

The rest of this paper is structured as follows. Section II describes related work on duplicate question detection. Section III presents the experimental common ground used on both reproduced studies, including data sets and evaluation measure. Section IV and V describes the empirical study focusing on the reproduction process, experiment execution, data analysis, and results. Section VI discusses our study validity. Finally, in Section VII, we draw conclusions and outline future work.

II. Related Work

We refer the reader to the comprehensive survey by Srba and Bielikova on Q&A websites [25]. The most closely related work includes techniques to retrieve similar questions and question classification. We discuss both categories bellow.

Question retrieval. Finding similar content on Q&A websites has a long history on academic research. Jeon et al. conduct a comparative study of four retrieval techniques (i.e., VSM, Okapi, language model, and translation-based model) [26]. Whereas the study took none question's structure into account, research in (semi) structured retrieval had indicated some improvements [27], [28]. Later, Theobald et al. combine stop-word antecedents with short chains of adjacent content terms [29]. Wu et al. resort to the Jaccard coefficient to measure similarities between two text segments in the pair of questions [30].

Other works explore answer's similarities to find equivalent questions. Hao and Agichtein propose an automatic patterns generation, which explores three methods of syntactic pattern: chunk-based phrase level, chunk-based lexical level, and tree-based incremental method [31]. Given a new question, it is compared to the set of available equivalent patterns. In case of a match, a prior answer is returned. Nie et al. present an algorithm for ranking answer candidates from all of the available answers for a new question. They rely on four types of features, named deep, topic-level, statistical, and user-centric [32].

We conclude from the existing question retrieval studies that textual features extracted from its structure can positively impact the performance of such task.

Question classification. A central mission of question classification research has been to understand the knowledge available on Question and Answer websites. And, specifically to Stack Overflow, how these knowledge can be used to assist software development [33]. Examples include types of question [34], askers main concerns [1], and questions topics [35]. Here, we focus on duplicates problem [14]–[18].

Several studies attempt to detect duplicates on Stack Overflow. Xu et al. predicted whether two questions, and their set of answers, posted on Stack Overflow are semantically related. They used a convolution neural network and reported that their analysis required 14 hours of CPU [16]. Fu and Menzies [36] extended Xu et al.'s work [16] and present a 84 times faster method based on SVM, advocating that the method to detect duplicates need to be carefully assessed with respect to its computational cost.

Language modeling also plays a role in duplicate detection. For instance, Mizobuchi and Takayama explored word-embedding techniques to deal with word ambiguities, which suggests accuracy improvements compared to Bag-of-Words approach [17]. Further evidence for the impact of word embedding stems from the recent study of Zhang et al. [18]. Their approach, *PCQADup*, rely on extracted features of question pairs. Word embedding (i.e., *word2vec*) is used to learn frequently co-occurred phrases pairs taken from duplicate question pairs. They reported a significant improve in results compared to *DupPredictor* and *Dupe*. We did not reproduce

PCQADup here, because the study was published recently. Future work should take it into consideration. Complementary, Zhang et al. analyzed the developed features of the prior work and suggest that combining vector similarity, relevance and association features gives the best performance for Stack Overflow duplicate detection [37].

This paper is based on two studies: one study published in 2015 by Zhang et al. [14] and another study published in 2016 by Ahasanuzzaman et al. [15]. The next sections present the reproduction of those works, and describing the adopted study process, study design, and study execution.

III. Preliminaries

In this section, we describe the common ground to evaluate and reproduce *DupPredictor* [14] and *Dupe* [15]. All the experiments were conducted over a server equipped with Intel® Xeon® at 2.4 GHz on 80 GB RAM, twelve cores, and 64-bit Linux Mint Cinnamon operating system.

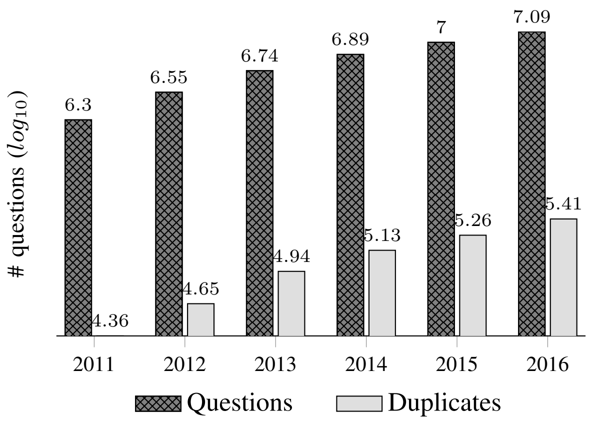


Fig. 2. Amount of questions and duplicates (non-master) per year in log_{10}.

A. Data Set

DupPredictor relied on the MSR'13 Challenge track data set [21], whereas *Dupe* used the MSR'15 one [22]. Note that both are officially released data dumps of Stack Overflow.

Our intent is to reproduce both works considering not only the original data sets but more recent ones. We could, at first, use the MSR'15 [22] data set in our replicates to reproduce both works since it is suppose to contain the ground truth of *DupPredictor* [14] and *Dupe* [15], this way not relying on MSR'13 data set [21] where duplicates were manually identified. However, in order to extend the results to more recent years, we would need more data sets containing more recent posts. Instead of working with multiple data sets, we opted to work with only one data set containing the ground truth of all considered years. This decision also avoided an extra effort (further detailed in section IV-B) about preprocessing multiple data sets. Hence, we adopted the last Stack Overflow official dump[7], which dates March 2017 and named this data set as our master data set. We used this data set for both reproductions and uniformly filtered posts by their creation dates, artificially building data sets for different years. We are aware that this master data set could not include posts deleted by users after the release of MSR'13 [21] and MSR'15 [22] data sets. Likewise, new duplicates discovered after the release of those data sets could influence the results. However we considered that this possible influence, if exists, would be minor compared to the original works.

[7]https://archive.org/details/stackexchange

As in *DupPredictor* [14] and *Dupe* [15], we define a master of a duplicate as a question asked first and the duplicate related to the master as the non-master. Figure 2 shows the number of questions per year, as well as the number of closed duplicated non-master questions from 2011 to 2016.

B. Evaluation Metric

We evaluate our reproductions by using the same metric adopted by *DupPredictor* [14] and *Dupe* [15], defined as:

$$recall\_rate@k = \frac{Ndetected}{Ntotal} \quad (1)$$

Ntotal is the total number of duplicate questions in the test set. *Ndetected* is the number of duplications found whose masters appear among the first *k* elements in the list of questions. *Recall-rate@k* is a measure that denotes the percentage of duplicated questions whose masters appear in the list among the first *k* elements.

IV. DupPredictorRep

In this section, we evaluate *DupPredictorRep*. The original work [14] also evaluates a variant of the approach without the use of Latent Dirichlet Allocation (LDA) – one topic modeling technique. Likewise, we also provide such variant called *DupPredictorRep-T*. We compare our results with *DupPredictor*. We also show how *DupPredictorRep-T* performs over different data sets.

A. Experimental Setup

The original data set used in *DupPredictor* is composed by posts before January 10, 2011. *DupPredictor* considers the first 2 million questions posted between July 2008 and September 2011. We artificially build this data set filtering posts by their creation date. Resulting in a number of 1,993,483 questions. This difference in the number of questions may be explained by the exclusion of questions after *DupPredictor* was built.

We assess *DupPredictorRep* over three more data sets. Each one is composed by posts where their creation dates are before January 10 of each year from 2012 to 2014.

In *DupPredictor*, 1,641 duplicated questions were found. They were manually inspected and the wrongly labeled as duplicates were removed, resulting in 1,528 duplicated questions. Instead of analyzing questions manually, we identify duplicated questions using the same approach as *Dupe* [15], which extracts the duplicated questions from the table *postlinks* and consider only those closed as duplicates. We select questions whose *closedDate* column value is not null in table *posts* and whose *id* is present in *postlinks* table. We consider this approach to be safer for identifying the duplicates because closing a question as duplicate in Stack Overflow requires an assessment by users with a considered degree of reputation. Even filtering by the creation date, the 2011 data set contains more than 1528 duplicated posts (actually more than 22 thousands) because the number of duplications tend to grow as new duplicates are constantly being discovered. We select the first 1,528 duplicated questions to simulate *DupPredictor*.

The tool WVTool [38] was used by DupPredictor to create word vector representations for the title, body and tag fields of each question. This tool also stems and removes stop words. Instead of using WVTool, we decided to re-implement the vector space model (VSM) [39] representation for these fields to avoid I/O overhead of WVTool. We tested our implementation and compared the results with WVTool. We input a set of questions into WVTool and to our implementation and carefully checked that the outputs of each tool matched in all cases. For titles and bodies, the weights of the terms are calculated using their term frequencies. For tags, weights are calculated based on the presence of the tag, or their binary frequencies.

For the stemming and stop word removal, Porter stemming algorithm [40] was used as did the original work, and the stop word list is the default provided by Lucene Framework.

B. Method

We divide the reproduction of *DupPredictor* in 4 steps. In Step 1, we preprocess the questions and prepare the data set for the duplicate detection. In Step 2, we generate the topics for the questions. In Step 3, we estimate the best weights for the composer component used to compare questions, and in Step 4 we perform the recall-rate calculation for *DupPredictorRep* and for *DupPredictorRep-T*.

Step 1: In the preprocessing step, *DupPredictor* collects the questions from Stack Overflow and perform stemming and stop words removal. However, questions available through the official dumps of Stack Overflow contain special characters in their bodies. Some of them are HTML tags, like "<*code*>", which contains code snippets, and others are special entities like *ampersand*, which is represented by "*&amp;*". As it is not clear how these kind of characters were handled in *DupPredictor*, we performed a test to identify the best approach to handle them. We considered two approaches:

1) We performed the stemming and removed the stop words over the titles and bodies of the questions. We called this approach *Raw Approach*.
2) We processed the title and body of each question. For both, we separated the content in tokens containing special characters and numbers (set 1) from the others (set 2). Over set 1 we removed HTML tags, special characters and punctuation symbols. Over the set 2, we performed stemming and removed stop words. Then we concatenated both sets and built the content of the title or body. We called this approach *Refined Approach*.

We tested these two approaches by calculating recall-rates over two data sets, each one generated from each approach.

TABLE I
RECALL-RATE VALUES FOR *Raw Approach* AND *Refined Approach*

Approach	Tag	R@100	R@50	R@20	R@10	R@5
Raw	html	17.60	15.18	11.93	9.98	8.10
	ruby	34.84	29.69	23.71	18.60	14.84
Refined	html	18.53	15.98	12.52	10.35	8.58
	ruby	36.64	32.31	25.82	20.87	17.06

DupPredictor uses a composer component to compares two questions and calculates scores for each question according to the Equation 2:

$$\begin{aligned} Composer_{nq}(oq) = {} & \alpha \cdot TitleSim_{nq}(oq) \\ & + \beta \cdot DesSim_{nq}(oq) \\ & + \gamma \cdot TopicSim_{nq}(oq) \\ & + \delta \cdot TagSim_{nq}(oq) \end{aligned} \tag{2}$$

In this component, the weights α, β, γ, and δ are associated to each of the subcomponents: *TitleSim, DesSim, TopicSim* and *TagSim*. Each subcomponent denotes the cosine similarity [39] value for titles, bodies, topics and tags respectively, between two questions. *DupPredictor* performed a series of experiments to obtain resonable weights to each subcomponent and concludes that in general when $\alpha > \beta > \delta > \gamma$ recall-rate@5, recall-rate@10 and recall-rate@20 are better. It estimated the best set of weights when $\alpha = 0.8, \beta = 0.51, \delta = 0.37$ and $\gamma = 0.01$.

We used these weights to test our two approaches. For this test, we used *DupPredictorRep-T* (γ = 0) because, besides the *TopicSim* weight is significantly smaller than the others, generating topics is computationally expensive. The data set used in this test was the one used in MSR Challenge 2015 [22]. We built two identical copies of this data set, where each approach ran over each data set. Then, for each data set we performed two tests to assess the recall-rates: one for posts whose tags contain *ruby*, and another one for posts whose tags contain *html*. We tested 7,717 questions for tag *html* and 1,940 questions for tag *ruby*. We compute recall-rate at different top-k: 100, 50, 20, 10 and 5. In all cases, the recall-rates for the *Refined Approach* are better than the ones for the *Raw Approach* as shown in Table I.

Thus, we discarded the two data sets used in this test and applied the *Refined Approach* on our master data set. After this process, we append the word "Duplicate" in the titles of the duplicated questions. We use the master data set for the next stages of *DupPredictorRep*.

Step 2: In this step, we use Mallet [41] toolkit to generate topic distribution for each question. For this, we build a folder containing text files representing all questions, each file containing the content of a question: the concatenation of title and body. Then we use this folder along with the number of topics as parameters to train topics. Next, the trained model generated by Mallet is used to generate the topics distribution for the questions. The output of Mallet is a text file containing these distributions where each line contains the identification of the question followed by the probabilities of the topics distribution, separated by spaces. We use the same number of topics as in *DupPredictor* (100). Finally, *DupPredictorRep* reads this file and load these distributions to each question.

Step 3: In this step, we reproduce weight estimation for α, β, γ and δ, used in the composer component. In the same way as *DupPredictor*, we use the first 20% of the 1,528 duplicate questions as training set to find weights for the composer component, ensuring that each question have its master question in the data set.

The purpose of this step is to find better weights for the composer component than the original ones. We performed a series of experiments, where we varied α, β, γ and δ and compute recall-rate for each set of values. Considering that calculating recall-rates is time consuming, specially for data sets containing millions of questions, we used the heuristic assumption of *DupPredictor* that $\alpha > \beta > \delta > \gamma$ produce better results to guide our generation. Our findings are described in Section IV-C.

Step 4: Here, we evaluate *DupPredictorRep* over the master data set. We build the test set composed by the first 1,528 duplicated questions. We excluded those ones used in the training set in previous step, and the ones whose masters were deleted by users. Resulting in a test set of 1,124 questions. We compare each of these 1,124 questions with the other 1,993,483 ones simulating the original data set.

We also calculate the recall-rate over new data sets containing more questions, artificially built by increasing the year of the creation date parameter. The purpose of this test is to assess the performance of *DupPredictorRep* as the number of questions in the data set increases. For these tests, we disable the LDA for the score calculations (we used *DupPredictorRep-T*). The reasons are the same as described in Section IV-B: the low influence on recall-rates and the high consumption of time to execute, specially in data sets where the number of questions grow considerably through the years as previously shown in Figure 2.

Besides the number of questions, the number of duplicate questions detected by users also had a significant growth. To make these set of tests feasible, we also limit the number of duplicate questions in the test set to 1,528. Out of these, we remove the first 20% that would represent the training set. Also, out of the remaining ones we remove those whose masters were deleted by users. The resulting test sets of the years 2012, 2013 and 2014 are 1,147, 1,165, and 1,172 duplicate questions respectively.

C. Results

We show the results achieved in Steps 3 and 4 .

Step 3 – Estimating Weights: 50 experiments were executed to estimate weights for α, β, γ and δ. Table II shows the best results for recall-rate@20, sorted in descending order. Weights in bold are those reported as the best in *DupPredictor*.

Step 4 – Calculating Recall-Rate: Table III shows the comparison of recall-rate values for *DupPredictor*, *DupPredictorRep* and *DupPredictorRep-T*. Table IV shows the recall-rate values obtained for *DupPredictorRep-T* against different sets.

D. Discussion

Despite the low influence on recall-rates, the use of topics was recommended in *DupPredictor*. Indeed, DupPredictorRep-T version compared to *DupPredictorRep* shows a decrease in terms of recall-rate of only 0.6%, 0.1% and 0.4% for recall-rate@20, recall-rate@10 and recall-rate@5, respectively.

Differently, the decrease in recall-rates is evident when the number of tested questions increases, as shown in Table IV. Compared to the 2011 data set, tests against 2012 and 2013 data sets present decays of 4.0% and 6.7% for the recall-rate@20 respectively while their number of questions increased 77.17% and 176.41%. However, the recall-rate@20 drops to 0.0% when the number of questions increased 286.95% in 2014. This weakness may be explained not only by the high number of unnecessary comparisons between two questions, as *DupPredictor* do not filter questions by tag, but also by the

TABLE II
RECALL-RATE@5, RECALL-RATE@10 AND RECALL-RATE@20 OF *DupPredictorRep* WITH DIFFERENT WEIGHTS FOR α, β, γ AND δ AND A TRAINING SET COMPOSED BY 285 DUPLICATED QUESTIONS (THE WEIGHTS IN BOLD ARE THE ONES FOUND IN *DupPredictor*)

Num	α	β	γ	δ	R@20	R@10	R@5
1	0.72	0.40	0.10	0.31	0.389	0.301	0.259
2	**0.80**	**0.51**	**0.01**	**0.37**	0.382	0.312	0.259
3	0.87	0.71	0.29	0.48	0.378	0.326	0.266
4	0.69	0.50	0.19	0.30	0.378	0.312	0.259
5	0.95	0.80	0.61	0.75	0.378	0.308	0.252
6	0.98	0.57	0.22	0.42	0.378	0.305	0.259
7	0.89	0.76	0.31	0.52	0.375	0.319	0.266
8	0.75	0.58	0.27	0.35	0.375	0.319	0.263
9	0.57	0.41	0.38	0.41	0.375	0.315	0.252
10	0.76	0.74	0.39	0.43	0.375	0.308	0.259
...	...	...	...	...	...	...	...
50	0.68	0.64	0.03	0.07	0.326	0.231	0.182

TABLE III
RECALL-RATE VALUES FOR *DupPredictor* (EXTRACTED FROM [14]) AND FOR THE REPLICATES: *DupPredictorRep* AND *DupPredictorRep-T* WITH α = 0.80, β = 0.51, γ = 0.01 AND δ = 0.37 AND A TEST SET COMPOSED BY 1124 DUPLICATED QUESTIONS

Approach	R@20	R@10	R@5
DupPredictor	0.638	0.533	0.423
DupPredictorRep	0.433	0.353	0.282
DupPredictorRep-T	0.427	0.352	0.278

TABLE IV
RECALL-RATE VALUES FOR *DupPredictorRep-T* IN DIFFERENT DATA SETS

Year	Test Set Size	R@100	R@50	R@20	R@10	R@5
2012	1,147	0.565	0.489	0.387	0.314	0.248
2013	1,165	0.533	0.453	0.360	0.287	0.236
2014	1,172	0.463	0.302	0.000	0.000	0.000

```
procedure ComputeRankingAndRecallRate($nonMasters, rateAt, tag$)
    $questions \leftarrow$ fetchQuestions($tag$)                          ▷ Fetch questions by tag
    $hits \leftarrow 0$
    $ranking$
    $index$                                                                ▷ Search engine index
    for each $question$ in $questions$ do
        $content \leftarrow$ concat($title, body, tags$)                  ▷ Question's structure concatenation
        $index$.add($content$)
    end for
    for each $nonMaster$ in $nonMasters$ do
        $hits \leftarrow index$.searchAt($rateAt$,$nonMaster$)            ▷ Fetch first $rateAt$ results from search engine
        for $i = 0$ to $hits$ do
            $questionPair \leftarrow (i, nonMaster)$
            $features \leftarrow$ extractFeatures($questionPair$)
            $probability \leftarrow$ logisticRegression($features$)       ▷ Compute the probability of duplication using the classifier
            $ranking \leftarrow$ tuple[$questionPair, probability$]
        end for
        sort($ranking$)                                                    ▷ Sort first 100 entries by descending probability
        if checkDuplication($ranking$) then                                ▷ Verify whether a duplicate was found
            $order \leftarrow$ position($ranking$)
        else
            return $-1$
        end if
        computeRecalRateAt($order$, $nonMasters.lenght$)
    end for
end procedure
```

Fig. 3. Pseudocode to rank questions and calculate recall-rates

used text retrieval model whose scoring function is based on the frequency of terms [15]. In this model, two non-duplicated questions may have a higher scoring function than two duplicated ones only because they have more common terms.

Concerning time efficiency, our findings show that the time spent by our test set, considering our hardware settings and using the best set of weights estimated by *DupPredictor*, is 125,223 seconds (~35 hours). Each question took an average of 111.40 seconds to be compared to the others. For the data sets of 2012, 2013, and 2014, the time spent to calculate the recall-rates was approximately 67, 111, and 205 hours respectively.

V. DupeRep

In this section we evaluate *DupeRep*. We run *DupeRep* over different data sets to compare the results against the ones reported by the original work [15].

A. Experimental Setup

The data set used in *Dupe* comprises questions created until September 2014 [22], where there are 130,888 non-master duplicates and 90,245 master questions. We use the same approach as in *Dupe* to extract duplicated questions and filter questions by creation date. We identify 134,261 non-master duplicates and 88,476 master questions. This difference may be explained by new questions marked as duplicates and questions that were deleted by users.

Besides posts created until September 2014, we also consider posts created until September 2015 and 2016. The purpose is to assess the performance of *Dupe* when the number of questions increases.

In *Dupe*, questions are filtered by tag. Six tags were considered: Java, C++, Python, Ruby, Html and Objective-C. We consider this same list of tags to assess *DupeRep* over the three data sets. Likewise, five features were used in a discriminative classifier: Cosine Similarity Value, Term Overlap, Entity Overlap, Entity Type Overlap, and *WordNet* [42] Similarity. In *Dupe*, Term Overlap showed lower recall-rates when combined with other features and was discarded in the early stages of experiments. Entity overlap, entity type overlap and *WordNet* similarity changed the recall-rates insignificantly and their use were not recommended by the original work. As the Cosine Similarity value show the best results alone, we only use this feature in the computation of recall-rates. Like in *Dupe*, we calculate the cosine similarity from each pair of questions for the following informations: title-title, title-body, body-title, body-body, tag-tag, title-tag, and code-code.

The classifier used in *Dupe* to rank questions is a Logistic Regression classifier, but the paper does not specify which implementation was used. We tried two well know implementations: *Stanford CoreNLP* [43] and *Weka* [44]. However, we show only the recall-rates for *Weka* because its implementation reported far superior results than *Stanford*'s one.

B. Method

The reproduction of *Dupe* is done in five steps. In Step 1, we preprocess the questions and prepare the data set for the duplicate detection. In Step 2, we extract the features from questions. In Step 3, we train the classifier. In Step 4, we calculate the recall-rates for *DupeRep* and *BM25Rep* (as the original work evaluates the *BM25* – one Information Retrieval technique). Finally, in Step 5, we calculate the recall-rates for *DupPredictorRep-T* filtering questions by tag to compare the results.

Step 1: In this step, we use the same data set used in *DupPredictorRep*. The *Refined Approach* is applied over this data set, besides stemming and stop words removal. Also, we extract code blocks from the bodies of questions and identify the synonyms of tags. We collected the set of synonyms through the *Stack Exchange Data Explorer*[8], because such information is not present in the official dump. Code blocks and synonyms of tags are stored during the preprocess of the questions of our master data set and are ignored by *DupPredictorRep* but considered in *DupeRep*.

Step 2: We extract features for the same number of duplicated and non-duplicated questions. For the duplicated questions, we fetch the closed duplicated non-master questions filtering by creation date and tag. For each question in this list, we identify its master questions. It may be the case that a question is related to more than one master question. For each master question we perform the following steps:

1) Check whether master question has not been deleted. If the master was deleted by users, the non-master question is discarded.
2) Check whether the master has answers. If the master question has no answer, the non-master question is discarded.
3) Build a pair of question containing the non-master and the master question, as well as label this pair as duplicated.
4) Extract features for the question pair.

For non-duplicated questions, we fetch questions randomly, limiting the number of questions to the number of pairs created previously. For each question in this list, we build a pair of questions in such a way that one is the question of the list and the other is randomly selected from the list of the closed duplicated non-master questions. Then we extract features for each pair and label the pair as non-duplicated.

Step 3: We fetch the lists of duplicated and non-duplicated pairs generated in the previous step. From the duplicated pairs list, we use 20% of pairs to a test list. Out of the 80% remaining pairs, we extract features and label them as "duplicated". From the non-duplicated pairs list, we use 80% of the pairs to extract features and label them as "non-duplicated". We then use all features, "duplicated" and "non-duplicated", to train our logistic regression classifier.

Step 4: The algorithm in Figure 3 represents a peseudocode for the calculation of recall-rates. The algorithm receives 3 parameters as input: the test set composed by non-masters questions, a parameter to filter the top *k* questions and the tag filter (e.g., JAVA). The output are recall-rate@20, recall-rate@10 and recall-rate@5. Firstly, the algorithm fetches questions by tag (line 2) and initializes variables representing the number of duplicates found (line 3), the ranking data structure (line 4), and the search engine index (line 5). For each question found by the tag filter, the algorithm concatenates title, body, and tag to build an index of this concatenation in *Lucene*'s indexing mechanism (lines 6~9). Next, in a external loop (lines 10~25), the algorithm iterates the non-master list of the questions in the test set. For each non-master question, it performs a search by the content of the question (line 11). The return, variable *hits*, holds the number of questions retrieved by BM25 [45] search mechanism, which order results by relevance. The variable *i* (line 12) represents the position of a question in the search. In a internal loop (lines 12~17), the algorithm first creates a pair of questions (line 13) containing the non-master question and a question returned by the searching at the position *i*. Next, it extracts features for the pair (line14), then a trained classifier is used to calculate the probability of features' values for the duplicated questions (line 15). The obtained probability, along with the pair, is stored in the ranking data structure (line 16). The algorithm then proceeds in line 18 where the first 100 entries of the ranking classifier map are cropped. These entries are the highest probabilities values, ordered in descending order. Next, the algorithm verify if among these entries relies a duplicate and if so, get its order (lines 19~23). The algorithms ends computing the recall-rates (line 24).

Step 5: In this step, we calculate recall-rates for *DupPredictorRep-T*, but instead of comparing each question with all the others, we compare only with those of the same tag group. The purpose of this test is twofold: (i) to assess the reproduction of *DupPredictor* performed in *Dupe* over the data set of 2014; (ii) to assess the performance of *DupPredictorRep-T* when questions are filtered by tag in more recent data sets. Besides the data set of 2014, we test *DupPredictorRep-T* using the same 6 tags as in *Dupe* in data sets of 2015 and 2016 extending the original comparisons.

C. Results

We run *DupeRep* and calculate the recall-rates under two perspectives. First, we consider the data set from 2014 used by *Dupe*. We list the findings of the original work [15] for *BM25*, *Dupe*, and *DupPredictor* and our findings: *BM25Rep*, *DupeRep*, and *DupPredictorRep-T* with tag filtering in Table V.

Second, we show results for recall-rates considering three data sets: 2014, 2015, and 2016. The results for *DupeRep* as well as for *DupPredictorRep-T* with tag filter over the three data sets are shown in Figure 4.

D. Discussion

In *DupeRep*, the average time to detect duplicates in data set of 2014 for tags Ruby, Python, C++, Objective-C, Html and Java was 18, 22, 21, 17, 23 and 18 seconds respectively. As opposed to *DupPredictor*, the idea of filtering questions by tag significantly reduces the cost for computing features,

[8]http://data.stackexchange.com/stackoverflow/query/new

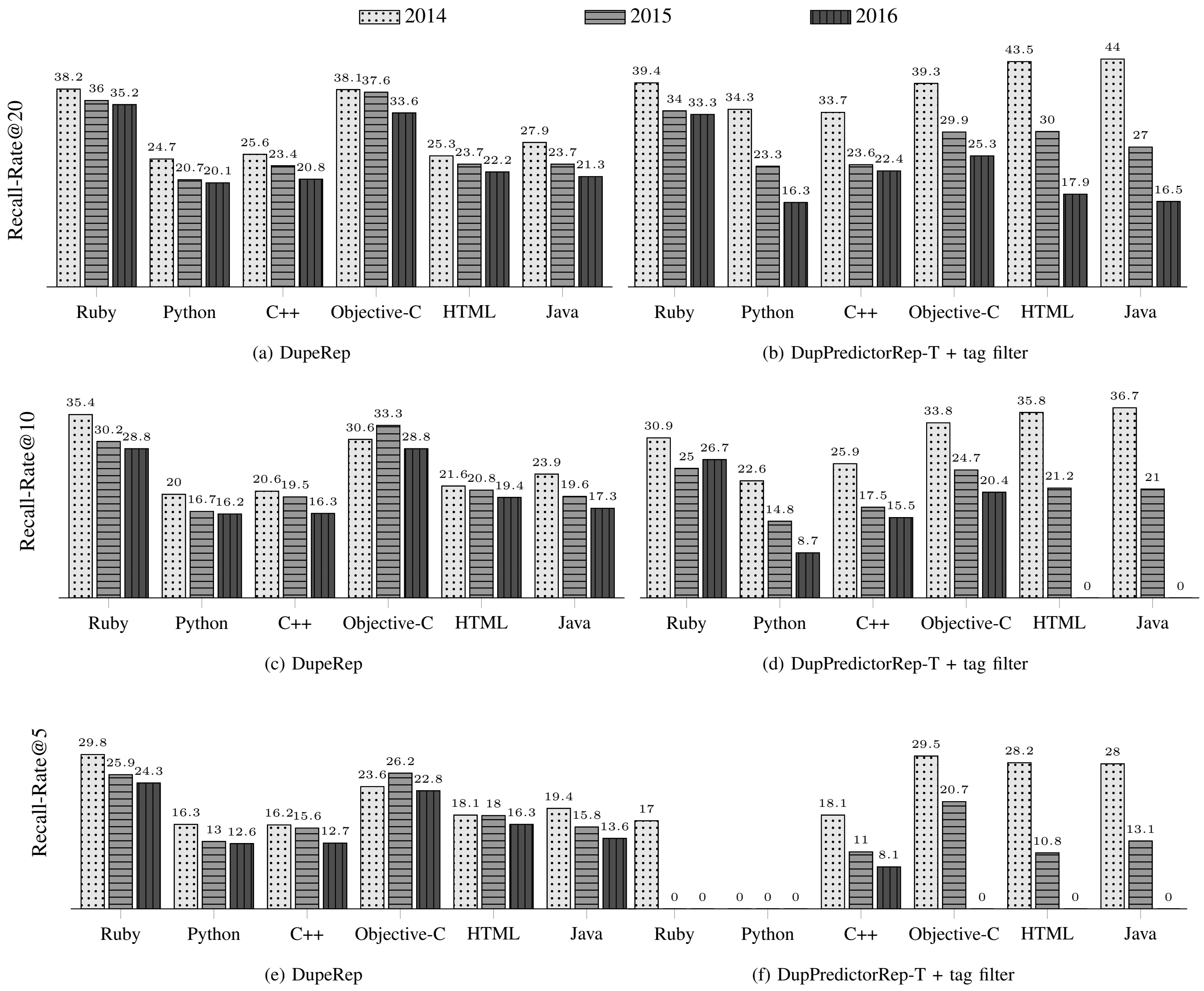


Fig. 4. Recall-Rate Values for *DupeRep* and *DupPredictorRep-T* + tag Filter in the Data Sets of 2014, 2015 and 2016.

as well as the time to calculate the recall-rates, once that the number of pairs being compared are much lower.

Our findings show that recall-rates for BM25 [45] search mechanism are considerably lower than the results reported in *Dupe*. For tags Ruby, Python, C++, Objective-C, Html and Java the recall-rate@20 absolute values are 32.99%, 26.59%, 23.71%, 22.94%, 20.21% and 25.79% lower respectively. Likewise, our findings for *DupeRep* show a decrease in recall-rate@20 for these tags of 27.85%, 28.5%, 24.26%, 18.25%, 24.93% and 25.07% in absolute values in comparison with *Dupe*.

When compared with *DupPredictor* implemented in the *Dupe* study for the data set of 2014, the values of *DupPredictorRep-T* with tag filter are higher for recall-rate@20 and recall-rate@10 for Objective-C, Html and Java. The highest difference was 9.56% for recall-rate@20. For the other tags, the most significant difference occurs in recall-rate@5 for tag Python where *DupPredictorRep-T* with tag filter was 0% while *DupPredictor* implemented in *Dupe* was 28.15%. *DupPredictorRep-T* with tag filtering also show a significant decrease in recall-rates through the years for all tag groups, specially for recall-rate@5 where in 2015 two out of six values resulted in 0.0% and in 2016 we found 0.0% for five out of the six analyzed tags. Our results reinforces the weakness of *DupPredictor*: its prediction power severely decreases when the number of assessed questions increases, mostly with regard to the hits in top positions. Moreover, we did not found the same results for *DupPredictor* as reported in *Dupe* study.

Notwithstanding, over the years *DupeRep* loses prediction power. But such performance decay is not as relevant as for *DupPredictorRep-T*. Considering the recall-rate@20 (in absolute values), the lowest decay occurred for Objective-c between 2014 and 2015 (0.5%) while the highest occurred for tag Java between 2014 and 2015 (4.15%). In *DupPredictorRep-*

TABLE V
RECALL-RATE VALUES FOR THE DIFFERENT TECHNIQUES IN DATA SET OF 2014.

Tag	Technique	R@20	R@10	R@5
Ruby	BM25	47.53	40.84	35.56
	BM25Rep	14.54	11.98	8.67
	Dupe	66.11	59.56	51.16
	DupeRep	38.26	35.45	29.84
	DupPredictor	39.00	37.00	33.00
	DupPredictorRep-T + tag	39.47	30.99	17.31
Python	BM25	36.09	30.18	26.10
	BM25Rep	9.50	7.40	6.23
	Dupe	53.22	45.41	37.20
	DupeRep	24.72	20.00	16.38
	DupPredictor	38.74	32.28	28.15
	DupPredictorRep-T + tag	34.38	22.61	0.00
C++	BM25	31.09	26.36	23.32
	BM25Rep	7.38	6.02	5.09
	Dupe	49.93	40.12	37.14
	DupeRep	25.67	20.63	16.26
	DupPredictor	35.00	28.00	26.00
	DupPredictorRep-T + tag	33.73	25.91	18.19
Objective-C	BM25	37.11	31.61	26.81
	BM25Rep	14.17	10.51	8.84
	Dupe	56.35	47.88	40.61
	DupeRep	38.10	30.64	23.62
	DupPredictor	31.00	28.00	26.00
	DupPredictorRep-T + tag	39.37	33.85	29.52
Html	BM25	31.22	27.11	23.94
	BM25Rep	11.01	9.20	8.47
	Dupe	50.23	39.45	37.14
	DupeRep	25.30	21.67	18.15
	DupPredictor	34.00	28.00	25.00
	DupPredictorRep-T + tag	43.56	35.89	28.21
Java	BM25	35.12	31.14	28.33
	BM25Rep	9.33	7.77	6.78
	Dupe	53.02	44.55	38.25
	DupeRep	27.95	23.90	19.44
	DupPredictor	41.00	35.00	30.00
	DupPredictorRep-T + tag	44.02	36.70	28.04

T, the lowest decay was 4.6% between 2012 and 2013 but 36% for 2013 to 2014. We confirm that *Dupe* has a more stable prediction power than *DupPredictor*, specially when the number of questions in increases.

Dupe compares its approach with *DupPredictor* against the data set of 2014. Our findings for this comparison are also different. Our results for *DupPredictorRep-T* combined with tag filtering surpass *DupeRep* results for recall-rate@20 under all tag groups. Also for recall-rate@10 in five out of the six analyzed tags. Despite *DupeRep* recall-rates tends to become higher than *DupPredictorRep-T* recall-rates for the data sets of 2015 and 2016, the presented results do not support the findings of *Dupe* regarding its comparison with *DupPredictor* for the proposed scenario.

VI. Threats to Validity

In this section, we discuss the threats to construct and internal validity. Also, what we adopted to mitigate them.

Construct validity is about the correct identification of measures adopted in the measurement procedure. In terms of evaluation bias, our results are not more biased than the original ones. We used the very same metric (i.e., recall-rate) to test and compare all reproduced approaches.

Internal validity refers to the factors that may have influenced our study. Instrumentality bias threatens any reproduction study, i.e., our design decisions may be interpreted in a different way from the original works. We did ask the first authors to provide some clarifications, but they did not answer back. We took care to either clearly define our algorithms or use implementations from the public domain (e.g., Apache Lucene, Weka). Also, all data and code used in this study are available on-line.

Another threat arises from the data set we used. Although the data came from Stack Overflow, we relied on the most recent available dump. So, we could broaden the results. To simulate the data used by *DupPredictor* and *Dupe*, we filtered Stack Overflow questions by the creation date. Hence, the questions taken to train and test our reproductions can differ from the original studies due to three main reasons. First, new questions were marked as duplicates after the original studies. Second, questions present in original studies may have been deleted by users ever since. And third, our samples are different from original studies. We mitigated the first reason by limiting the number of duplicates to the same used in the original studies. For the second, we assured that questions in test sets had their masters in our data set, removing the ones deleted by users. Lastly, we generated our samples following the same rules from the original studies. Despite our effort to reproduce both approaches exactly the same way, the random nature of samples generation may influence the results.

VII. Conclusion and Future Work

In this work, we presented an empirical study aimed at analyzing duplicate question detection in Stack Overflow. For this purpose, we conducted an independent reproduction of two previous approaches, *DupPredictor* [14] and *Dupe* [15].

Our reproduction produced lower recall-rates compared to the original studies. Although it is still not clear why our results differ from the original ones because we have different implementions and different datasets, we make a replication package public available to enable other researchers to repeat, improve, or refute our results. Moreover, we observed a performance decrease with the reproduction of those two approaches over time, as the number of question increases.

Further reproduction studies, with different settings, could extend the collected results. A more thorough investigation on how to stabilize duplicate question detection as the number of questions increase could also be conducted.

Acknowledgment

We would like to thank the Brazilian research agencies CNPq, CAPES, and FAPEMIG. K. Paixão is supported by a CAPES Ph.D. scholarship.

References

[1] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (nier track)," in *33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 804–807.

[2] F. Chen and S. Kim, "Crowd debugging," in *10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2015, pp. 320–332.

[3] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Prompter," *EMPIR SOFTW ENG*, vol. 21, no. 5, pp. 2190–2231, Oct 2016.

[4] S. Azad, P. C. Rigby, and L. Guerrouj, "Generating API call rules from version history and Stack Overflow posts," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 4, pp. 29:1–29:22, Jan. 2017.

[5] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social Q&A sites are changing knowledge sharing in open source software communities," in *17th Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, 2014, pp. 342–354.

[6] M. A. Storey, A. Zagalsky, F. F. Filho, L. Singer, and D. M. German, "How social and communication channels shape and challenge a participatory culture in software development," *IEEE Trans. Softw. Eng.*, vol. 43, no. 2, pp. 185–204, Feb 2017.

[7] J. Surowiecki, *The wisdom of crowds*. Anchor, 2005.

[8] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov, "Using and asking: APIs used in the Android market and asked about in StackOverflow," in *5th International Conference on Social Informatics (SocInfo)*. Cham: Springer, 2013, pp. 405–418.

[9] E. C. Campos, L. B. L. de Souza, and M. d. A. Maia, "Searching crowd knowledge to recommend solutions for API usage tasks," *J SOFTW EVOL PROC*, vol. 28, no. 10, pp. 863–892, 2016.

[10] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from Stack Overflow," in *38th International Conference on Software Engineering (ICSE)*, 2016, pp. 392–403.

[11] L. An, O. Mlouki, F. Khomh, and G. Antoniol, "Stack Overflow: A code laundering platform?" in *24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 283–293.

[12] R. Abdalkareem, E. Shihab, and J. Rilling, "What do developers use the crowd for? a study using Stack Overflow," *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017.

[13] I. Srba and M. Bielikova, "Why is Stack Overflow failing? preserving sustainability in community question answering," *IEEE Software*, vol. 33, no. 4, pp. 80–89, July 2016.

[14] Y. Zhang, D. Lo, X. Xia, and J.-L. Sun, "Multi-factor duplicate question detection in Stack Overflow," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 981–997, Sep 2015.

[15] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Mining duplicate questions in Stack Overflow," in *13th International Conference on Mining Software Repositories (MSR)*, 2016, pp. 402–412.

[16] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *31st International Conference on Automated Software Engineering (ASE)*, 2016, pp. 51–62.

[17] Y. Mizobuchi and K. Takayama, "Two improvements to detect duplicates in Stack Overflow," in *24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 563–564.

[18] W. E. Zhang, Q. Z. Sheng, J. H. Lau, and E. Abebe, "Detecting duplicate posts in programming qa communities via latent semantics and association rules," in *26th International Conference on World Wide Web (WWW)*, Geneva, Switzerland, 2017, pp. 1221–1229.

[19] A. Pal, R. Farzan, J. A. Konstan, and R. E. Kraut, "Early detection of potential experts in question answering communities," in *19th International Conference on User Modeling, Adaption and Personalization (UMAP)*. Berlin, Heidelberg: Springer, 2011, pp. 231–242.

[20] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *E.A. on Human Factors in Computing Systems (CHI EA)*, 2011, pp. 2425–2428.

[21] A. Bacchelli, "Mining challenge 2013: Stack overflow," in *10th International Conference on Mining Software Repositories (MSR)*, 2013.

[22] A. T. T. Ying, "Mining challenge 2015: Comparing and combining different information sources on the Stack Overflow data set," in *12th International Conference on Mining Software Repositories (MSR)*, 2015.

[23] G. Robles, "Replicating MSR: A study of the potential replicability of papers published in the mining software repositories proceedings," in *7th Work. Conf. on Mining Software Repositories (MSR)*, 2010, pp. 171–180.

[24] S. Amann, S. Beyer, K. Kevic, and H. Gall, "Software mining studies: Goals, approaches, artifacts, and replicability," in *Software Engineering: Intl. Summer Schools, LASER 2013-2014, Italy, Rev. Tutorial Lectures*, B. Meyer and M. Nordio, Eds. Cham: Springer, 2015, pp. 121–158.

[25] I. Srba and M. Bielikova, "A comprehensive survey and classification of approaches for community question answering," *ACM Trans. Web*, vol. 10, no. 3, pp. 18:1–18:63, Aug. 2016.

[26] J. Jeon, W. B. Croft, and J. H. Lee, "Finding similar questions in large question and answer archives," in *14th International Conference on Information and Knowledge Management (CIKM)*, 2005, pp. 84–90.

[27] D. Ravichandran and E. Hovy, "Learning surface text patterns for a question answering system," in *40th Annual Meeting on Association for Computational Linguistics (ACL)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 41–47.

[28] M. I. M. Azevedo, K. V. R. Paixão, and D. V. C. Pereira, "Processing heterogeneous collections in xml information retrieval," in *Advances in XML Information Retrieval and Evaluation: 4th INEX, Dagstuhl Castle, Germany*. Springer, 2006, pp. 388–397.

[29] M. Theobald, J. Siddharth, and A. Paepcke, "Spotsigs: Robust and efficient near duplicate detection in large web collections," in *31st Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, 2008, pp. 563–570.

[30] Y. Wu, Q. Zhang, and X. Huang, "Efficient near-duplicate detection for Q&A forum," in *5th International Joint Conference on Natural Language Processing (IJCNLP), Chiang Mai, Thailand*, 2011, pp. 1001–1009.

[31] T. Hao and E. Agichtein, "Finding similar questions in collaborative question answering archives: toward bootstrapping-based equivalent pattern learning," *Inf Retrieval*, vol. 15, no. 3, pp. 332–353, Jun 2012.

[32] L. Nie, X. Wei, D. Zhang, X. Wang, Z. Gao, and Y. Yang, "Data-driven answer selection in community qa systems," *IEEE Trans. Knowl. Data Eng*, vol. 29, no. 6, pp. 1186–1198, June 2017.

[33] F. M. Delfim, K. V. R. Paixão, D. Cassou, and M. de Almeida Maia, "Redocumenting APIs with crowd knowledge: a coverage analysis based on question types," *J. Braz. Comput. Soc*, vol. 22, no. 1, p. 9, 2016.

[34] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *28th Intl. Conf. on Software Maintenance (ICSM)*, 2012, pp. 25–34.

[35] L. B. L. de Souza, E. C. Campos, and M. d. A. Maia, "Ranking crowd knowledge to assist software development," in *22nd International Conference on Program Comprehension (ICPC)*, 2014, pp. 72–82.

[36] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2017, pp. 49–60.

[37] W. E. Zhang, Q. Z. Sheng, Y. Shu, and V. K. Nguyen, "Feature analysis for duplicate detection in programming QA communities," in *13th International Conference on Advanced Data Mining and Applications (ADMA)*. Cham: Springer, 2017, pp. 623–638.

[38] M. Wurst, "The word vector tool user guide operator reference developer tutorial," 2007. [Online]. Available: http://sourceforge.net/projects/wvtool/

[39] R. B. Yates and B. R. Neto, "Modern information retrieval: the concepts and technology behind search," *Addison-Wesley Professional*, 2011.

[40] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[41] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002.

[42] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995.

[43] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60.

[44] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[45] S. E. Robertson and S. Walker, "Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval," in *17th International Conference on Research and Development in Information Retrieval (SIGIR)*. Springer, 1994, pp. 232–241.