Duplicate Detection in Programming Question Answering Communities

WEI EMMA ZHANG and QUAN Z. SHENG, Macquarie University
JEY HAN LAU, The University of Melbourne and IBM Research Australia
ERMYAS ABEBE, IBM Research Australia
WENJIE RUAN, University of Oxford

Community-based Question Answering (CQA) websites are attracting increasing numbers of users and contributors in recent years. However, duplicate questions frequently occur in CQA websites and are currently manually identified by the moderators. Automatic duplicate detection, on one hand, alleviates this laborious effort for moderators before taking close actions, and, on the other hand, helps question issuers quickly find answers. A number of studies have looked into related problems, but very limited works target Duplicate Detection in Programming CQA (PCQA), a branch of CQA that is dedicated to programmers. Existing works framed the task as a supervised learning problem on the question pairs and relied on only textual features. Moreover, the issue of selecting candidate duplicates from large volumes of historical questions is often unaddressed. To tackle these issues, we model duplicate detection as a two-stage "ranking-classification" problem over question pairs. In the first stage, we rank the historical questions according to their similarities to the newly issued question and select the top ranked ones as candidates to reduce the search space. In the second stage, we develop novel features that capture both textual similarity and latent semantics on question pairs, leveraging techniques in deep learning and information retrieval literature. Experiments on real-world questions about multiple programming languages demonstrate that our method works very well; in some cases, up to 25% improvement compared to the state-of-the-art benchmarks.

CCS Concepts: • **Information systems → Near-duplicate and plagiarism detection**; **Question answering**; **Association rules**;

Additional Key Words and Phrases: Community-based question answering, question quality, classification, latent semantics, association rules

ACM Reference format:
Wei Emma Zhang, Quan Z. Sheng, Jey Han Lau, Ermyas Abebe, and Wenjie Ruan. 2018. Duplicate Detection in Programming Question Answering Communities. *ACM Trans. Internet Technol.* 18, 3, Article 37 (April 2018), 21 pages.
https://doi.org/10.1145/3169795

Q. Z. Sheng's work has been partially supported by Australian Research Council (ARC) Future Fellowship Grant FT140101247.

Authors' addresses: W. E. Zhang and Q. Z. Sheng, Department of Computing, Macquarie University, NSW 2109, Australia; emails: {w.zhang, michael.sheng}@mq.edu.au; J. H. Lau, Department of Computing and Information Systems, The University of Melbourne and IBM Research Australia, VIC 3010 Australia; email: jeyhan.lau@gmail.com; E. Abebe, IBM Research Australia, 204 Lygon Street, VIC 3053, Australia; email: etabebe@au1.ibm.com; W. J. Ruan, Department of Computer Science, University of Oxford, Parks Road, Oxford, OX1 3QD, UK; email: wenjie.ruan@cs.ox.ac.uk.

https://doi.org/10.1145/3169795

1 INTRODUCTION

There has been an increase in the popularity of Community-based Question Answering (CQA) websites such as Quora[1], Yahoo! Answers,[2] and Stack Exchange[3] on the Internet. The reason is that CQA websites are becoming a promising alternative to traditional web search to provide answers that are subjective, open-ended, and with expert opinions. To cater to the multitude of interests in its community, many CQA websites hold a set of subforums that focus on a particular topic or theme. For example, Stack Exchange[4] spans different themes like Science ("Mathematics," "Physics"), Life ("Home Improvement," "Travel") and Technology ("Stack Overflow," "Ask Ubuntu").

Stack Overflow, a Programming Community-based Question Answering (PCQA) site, is a sub-domain in Stack Exchange created for programming-related questions. Despite detailed guidelines on posting ethics, a large number of created questions are poor in quality [13]. Duplicate questions—questions that were previously created and answered—are a frequent occurrence even though users are reminded to search the forum before creating a new post. To reduce the number of duplicate questions, Stack Overflow encourages reputable users to manually mark duplicate questions. Figure 1 illustrates the Stack Overflow workflow for marking duplicate questions, where enough votes from reputable users are required. This approach is laborious, but, more seriously, a large number of duplicate questions remain undetected for long time. As reported in Ahasanuzzaman et al. [1], more than 65% of duplicate questions take at least one day to be closed, and a large proportion of duplicate questions are closed more than one year later. Therefore, a high-quality automatic duplicate detection system is required to considerably improve user experience: For inexperienced users creating a new question, it can suggest a related post before posting and possibly retrieve answers immediately; for experienced users, it can suggest potential duplicate posts for manual verification.

Very limited studies have explored question duplication for the PCQA domain [1, 43]. The existing works adopted the duplicate mining (or similar tasks) methods from the CQA domain that framed the task as a classification or prediction task and relied on a number of extracted features to train a model [8, 9, 33, 37, 38, 44]. They differed from the CQA methods in extracting additional or different features. However, two main issues exist in these methods. First, as PCQA questions often contain source code from programming languages which are linguistically very different from natural languages, features that only consider the natural language text in these methods may miss important indicator features of PCQA questions. In addition, they consider only syntactical and lexical features that are not necessarily able to reveal the latent semantics of questions. Second, as the number of questions posted on PCQA websites increases dramatically, rendering the identification of possible duplicate questions from large volume of historical questions becomes a tedious task. To the best of our knowledge, only one work [1] very briefly addressed this issue by directly applying one candidate selection method. No existing work used different methods to perform candidate selection and provide performance comparisons.

In this article, we seek to improve on the benchmark performance by developing a two-stage "ranking-classification" detection method that addresses the just-discussed issues. Before starting the detection process, our methodology performs preprocessing tailored to the programming-related questions and aiming to maximally obtain information from questions. In the first stage of duplicate detection, our methodology selects candidate duplicates given a new question by

[1] https://www.quora.com/.
[2] https://answers.yahoo.com/.
[3] http://stackexchange.com/.
[4] https://stackexchange.com/.

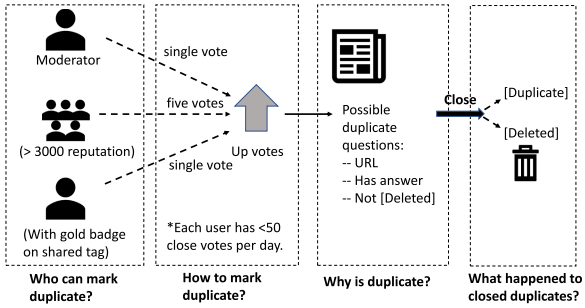


Fig. 1. Stack overflow duplicate detection workflow.

ranking all historical questions and selecting the top ranked ones. In addition to using tags to prune the questions in different domains, we propose to adopt three empirically effective ranking algorithms to rank all the historical questions according to their similarities to the new question. The three algorithms are query likelihood using language models with Jelinek-Mercer smoothing (LMJM) [21], query likelihood using language models with Bayesian smoothing; and adopting Dirichlet priors (LMDIR) [40] and BM25 [32]. Last, we develop a ranking algorithm based on the topical similarities learned from the topic model. It is worth noting that the ranking within the candidates does not matter as we only examine whether the possible duplicates are selected.

The second stage of our methodology is about detecting duplicate questions from the selected candidates. The method follows the same approach from previous works by framing the duplication detection task as a supervised classification problem. Henceforth, we refer to our system as `DupDetector`. Given a newly issued question, `DupDetector` pairs the question with each of the candidate questions. Then it employs classification models to predict whether the question pair is duplicate (positive) or nonduplicate (negative). To train the classifiers, `DupDetector` generates three types of features for each question pair. The *vector similarity feature* represents questions as continuous vectors in a high-dimensional space by adopting `doc2vec` [24]. This feature effectively captures the semantics of questions. The *relevance feature* adapts the query-document relevance measurements used in our ranking stage. Except for the introduced LMJM, LMDRI, and BM25, we also adapt Divergence from Randomness Similarity (DFRS) [3] and Information-Based Similarity (IBS) [11]. An *association feature* is generated on mined association phrases (i.e., pairs of phrases that co-occur frequently in known duplicate questions) and lexical features. The idea of using association phrases is adopted from ranking generated queries in curated Knowledge Base Question Answering (KBQA) [5] and measuring the quality of question paraphrases in open KBQA [39].

To summarize, the main contributions of this article are:

(1) **Candidate selection before duplicate detection**: We select candidate duplicate questions from historical questions by ranking them by their relevance to the newly posted question before performing duplicate detection. This step reduces the search space of the possible duplicate questions and is beneficial especially when encountering a large corpus of historical questions.
(2) **Novel features for duplicate question detection**: We develop three categories of novel features that consider both textual (association feature, relevance feature) and latent (vector similarity feature) information to facilitate duplicate question detection.
(3) **Association pairs for PCQA**: We mine more than 130K association pairs from known duplicate questions in Stack Overflow. The data contain phrase pairs that frequently occur in duplicate question pairs and are domain-specific to PCQA.

(4) **Extensive experimental evaluation**: We provide extensive analysis on the behavior of different features independently and the effectiveness of their combinations in duplicate detection. We compare the impact of candidate selection methods. Results on a real-world dataset suggest that DupDetector outperforms state-of-the-art benchmark by 4% to 25% in terms of recall for duplicate detection on various programming languages.

This article substantially improves our previous work presented at the WWW conference in 2017 [42]. First, we add the candidate selection stage to reduce the search space of duplicate questions and frame the detection problem as a two-stage task. Second, we develop new association and relevance features to replace the topical feature and association score feature and improve the vector similarity feature proposed in Zhang et al. [42]. Third, we perform much more extensive experiments on both classification performance evaluation and the performance of the DupDetector system.

The rest of the article is organized as follows. In Section 2, we overview our system DupDetector, followed by the technical details of the two stages in Section 3 and Section 4. We report experimental results in Section 5. In Section 6, we review the related works. Finally, we conclude and discuss some future research directions in Section 7.

2 OVERVIEW OF THE DUPDETECTOR SYSTEM

In this section, we overview our system DupDetector, which detects duplicate questions by framing the task as a two-stage "ranking-classification" problem. The technical details of the two stages are given in separate sections (Section 3 and Section 4).

To identify whether a new question has duplicates in existing questions, DupDetector first selects questions that could be potential duplicates, then pairs the selected questions to the new question and performs classification on question pairs. Specifically, a ranking-based candidate selection is performed which selects potential duplicate questions from the existing questions given a newly issued question. The intuition of this component is to reduce the search space for potential duplicates because constructing features (to be used in classification) for the new question with each of the historical questions is a time-consuming task. We will further discuss our proposed candidate selection methods in Section 3.

After obtaining the selected candidates, DupDetector pairs these candidates to the new question and then extracts features from the question pairs. The classifier is trained using features extracted from existing labeled duplicates and corresponding nonduplicate question pairs; these question pairs are considered as ground truth. The training process is not activated each time a new question is issued, but performs periodically and independently of the detection process to reflect the new questions and update features. Then, for the question pairs constructed by the selected candidates and newly issued question, DupDetector fits the features extracted from these pairs into the trained classifier to perform classification. All the predicted duplicate questions are listed if multiple duplicates exist. If no duplicates are found, the question will be posted to PCQA communities as a normal post. As feature modeling is a key component in classification that largely affects classification performance or duplicate detection performance, we propose novel features that capture both syntactic and latent semantic relations between the paired questions; a detailed discussion follows in Section 4.

3 CANDIDATE SELECTION

As the number of questions increase, there is a huge amount of historical questions in PCQA communities. Duplicate detection in these questions becomes a time-consuming task. Before DupDetector performs classification on question pairs (e.g., (m, t)), we first select some candidate

questions m to be paired with the newly issued question t. This step reduces the search space of the duplicated question t. Specifically, we propose four methods to implement candidate selection and discuss them in this section. Selection with tags (Section 3.1) is the basic method that all three selection methods perform. Query likelihood algorithms (Section 3.2) are adapted, inspired by the ranking systems used in search engines. BM25 is a well-known and effective ranking algorithm for information retrieval tasks, and we adapt it to our candidate selection task (Section 3.3). Finally, we discuss our proposed method that utilizes a topic model to select candidate questions (Section 3.4).

3.1 Candidate Selection with Tags

Tags are mandatory inputs when posting a new question on many PCQA forums (e.g., all subforums of StackExchange). Therefore, they are reliable indicators of the topics the question belongs to. We conduct a naive filtering approach on possible m to filter out questions that belong to a different programming language or technique using tags. Specifically, we prune existing questions that have no common tags with t, thus narrowing the search space for candidate duplicate questions considerably. We take this selection method as a prerequisite for other selection methods.

3.2 Candidate Selection with Query Likelihood Model

Upon naive selection using tags, we propose a further selection based on the query likelihood algorithms. The basic idea of the query likelihood is to estimate a language model for each document in a collection and then rank them by the likelihood of the query according to the estimated language model [40]. A core problem in language model estimation is *smoothing*, which assigns non-zero probability to unseen words, aiming to achieve higher accuracy.

Query likelihood is utilized in search engines as a "quick-ranking" of possible documents to an issued query before performing a more complex "re-ranking" step which outputs the final ranked list of documents given a query. We adopt this idea and propose to rank all the historical questions and choose the top ranked questions as candidates to further perform the classification task. To apply this query likelihood-based ranking in the candidate selection in our work, we consider each existing question as a document and turn the "query likelihood" into "question likelihood," which examines the likelihood between existing question m and new question t. The question likelihood is defined using log likelihood, as follows by adapting equations in Zhai et al. [40]:

$$\log P(\mathbf{t}|\mathbf{m}) = \sum_{w \in \mathbf{t}} \log \frac{P(w|\mathbf{m})}{\alpha P(w|Q)} + n \log \alpha + \sum_{w \in \mathbf{t}} \log P(w|Q), \tag{1}$$

where w is a word in the new question t, n is the length of the query, α is related to the smoothing parameter, and Q is the collection of the questions. When performing ranking, $\sum_{w \in \mathbf{t}} \log P(w|Q)$ can be ignored as it is independent of the question m.

We adopt Jelinek-Mercer smoothing [21], and the language model is defined as follows:

$$\begin{aligned} P(w|\mathbf{m}) &= (1-\lambda) P_{ML}(w|\mathbf{m}) + \lambda P_{ML}(w|Q), \\ P_{ML}(w|\mathbf{m}) &= \frac{tf(w,\mathbf{m})}{\sum_{w' \in \mathbf{m}} tf(w',\mathbf{m})}, \\ P_{ML}(w|Q) &= \frac{tf(w,Q)}{\sum_{w' \in Q} tf(w',Q)}, \end{aligned} \tag{2}$$

where $P_{ML}(w|\mathbf{m})$ is the *Maximum Likelihood* estimate of word w in existing question $\mathbf{m}$, $P_{ML}(w|Q)$ is the *Maximum Likelihood* estimate of word w in the question collection Q, $tf(w,\mathbf{m})$ is the frequency of word w in $\mathbf{m}$, and λ is the smoothing parameter ($\alpha = \lambda$).

We also apply Bayesian smoothing using Dirichlet priors [40], and the language model is defined as follows:

$$P(w|\mathbf{m}) = \frac{tf(w,\mathbf{m}) + \mu P(w|Q)}{\sum_{w' \in \mathbf{m}} tf(w',\mathbf{m}) + \mu}, \tag{3}$$

where the Dirichlet parameters are $\mu P(w_1|Q), \mu P(w_2|Q), \ldots, \mu P(w_n|Q)$, n is the total number of unique words in Q, μ is the smoothing parameter, and $\alpha = \frac{\mu}{\sum_{w' \in \mathbf{m}} tf(w',\mathbf{m}) + \mu}$.

We adopt the parameter suggestions provided by empirical studies on smoothing methods for language models given by Zhai et al. [40].

3.3 Candidate Selection with BM25

BM25 [32] is another widely used and empirically well-performing ranking algorithm. It is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document. To adapt BM25 to our task to score candidate historical question m given a new question t, we use the following equation [1]:

$$BM25(t,m) = \sum_{w \in (\mathbf{t} \cap \mathbf{m})} tf(w,\mathbf{t}) \frac{(k+1)tf(w,\mathbf{m})}{tf(w,\mathbf{m}) + k(1 - b + \frac{b|m|}{avgl})} log \frac{n+1}{df(w)}, \tag{4}$$

where $tf(w,\mathbf{m})$ is the frequency of word w in the master question $\mathbf{m}$, $avgl$ is the average length of questions, $df(w)$ is the number of questions word w appear, n is the total number of questions, and b and k are free parameters.

3.4 Candidate Selection with Topics

In this section, we discuss our proposed method that ranks candidate duplicate questions using topical similarity, which is obtained by the topic model. Latent Dirichlet Allocation (LDA) [6] is a well-known implementation of the topic model that extracts topics unsupervisedly from document collections. It posits that each document is a mixture of a small number of topics, and each word's creation is attributable to one of the document's topic. The outputs of the algorithm are two distributions: the document-topic multinomials and topic-word multinomials. The topic distribution (i.e., document-topic multinomials) for a document learned by LDA constitutes its topical vector. LDA does not work well for short texts and, as such, may not be directly applicable to PCQA posts. To learn topics for our dataset, we adopt a modified LDA [25], which is designed for extracting topics from short texts. To apply LDA, we consider each question as a document. We learn a topic model for the concatenation of question titles and body contents. Then we compute cosine similarity between topical vectors of a question pair (m,t). Question pairs are ranked according to the computed topical similarities, and we choose the top-ranked pairs as the candidate duplicate pairs for further examination.

4 FEATURE MODELING

Upon obtaining candidate questions, we pair them with the new question and generate features from the pair. Effective features are developed to classify the pairs as duplicate or nonduplicate. Given a question pair (m,t), we develop three groups of features, as listed in Table 1. The features include vector similarity (Section 4.1) and relevance features (Section 4.2), which are computed on the full sentence of m and t, and an association feature (Section 4.3), which leverages the mined association pairs and lexical features on spans of m and t.

Table 1. Feature Generation in DupDetector. (m, t) Is a Pair of Questions; s_m, s_t Are Spans in m and t, Respectively

Vector similarity feature*.	
▷ cosine similarity of vectors on m and t using doc2vec.	
Relevance features*.	
▷ $LM_{jm}(m \mapsto t)$, question likelihood using Jelinek-Mercer smoothing.	▷ $BM25(m \mapsto t)$, BM25 value of the (m, t) pair.
▷ $LM_{dir}(m \mapsto t)$, question likelihood using Dirichlet prior smoothing.	▷ $IBS(m \mapsto t)$, information-based similarity.
▷ $DRFS(m \mapsto t)$, divergence from randomness similarity.	
Association feature [5, 39].	
▷ count of $\langle lemma(s_m), lemma(s_t)\rangle$ if $\langle lemma(s_m), lemma(s_t)\rangle \in (\mathcal{A})$, 0 otherwise. ($\mathcal{A}$ is the set of mined association pairs)	
▷ $lemma(s_m) \wedge lemma(s_t)$. $lemma(w)$ is the lemmatized word of w.	▷ $pos(s_m) \wedge pos(s_t)$. $pos(w)$ is the POS tag of w.
▷ $lemma(s_m)$ and $lemma(s_t)$ are synonyms?	▷ $lemma(s_m)$ and $lemma(s_t)$ are WordNet derivations?

*Vector feature and other features are computed on m and t's title, body and, the concatenation of the both.

4.1 Vector Similarity Feature

The vector similarity feature is computed on the vector representation of master question m and target question t. A conventional approach to vectorize text is to compute `tf-idf` scores for documents. Each question is considered as a document, and we generate a `tf-idf` vector. Cosine similarity for a question pair is then computed to serve as the vector similarity feature, which is considered as the baseline in this work.

Plenty of other methods could map text to high-dimensional space, among which neural methods for learning word embeddings/vectors have seen many successes for a range of NLP tasks. `word2vec`, a seminal work proposed by Mikolov et al. [28], is an efficient neural architecture to learn word embeddings via negative sampling using a large corpora of text. Paragraph vectors, or `doc2vec`, was then introduced to extend `word2vec` to learn embeddings for word sequences [24]. `doc2vec` is agnostic to the granularity of the word sequence: It could learn embeddings for a sentence, paragraph, or document. Two implementations of `doc2vec` were originally proposed: `dbow` and `dmpv`. The work in Lau and Baldwin [23] evaluated `dbow` and `dmpv` on a range of extrinsic tasks and found that the simpler `dbow` trains faster and outperforms `dmpv`. We therefore use the `dbow` architecture for all experiments involving `doc2vec` in this article. Given the question pair (m, t), we apply `doc2vec` on the title, body content, tags, concatenation of titles, and body contents (title+body) and the concatenation of title, body, and tags (title+body+tag) on both master question m and target question t. Thus, we train five `doc2vec` models, one for each type of vector, and obtain 10 vectors for each question pair, namely t_t, t_m, b_t, b_m, ta_t, ta_m, tb_t, tb_m, $tbta_t$, and $tbta_m$ in Figure 2. Cosine similarities on (t_t, t_m), (b_t, b_m), (ta_t, ta_m), (tb_t, tb_m), and $(tbta_t, tbta_m)$ constitute vector similarity features. The vector feature is generated according to the first part of Table 1.

4.2 Relevance Feature

The relevance feature reflects the relevance between question m and t in a question pair (m, t). The relevance feature is generated according to the second part of Table 1. Question likelihood

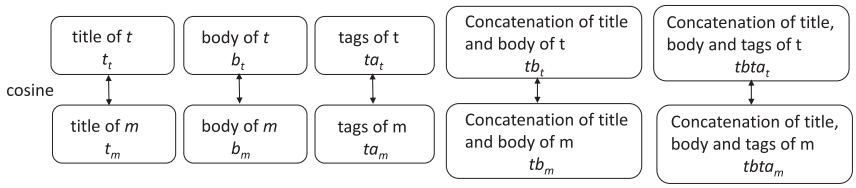


Fig. 2. Vector similarities computation. The subscript $_t$ and $_m$ denote target question t and master question m, respectively.

using a language model with Jelinek-Mercer smoothing and Bayesian smoothing are computed according to Equations (1) to (3), and the likelihood value serves as two features referred to as $\mathbf{lm_{jm}}$ and $\mathbf{lm_{dir}}$. BM25 is computed using Equation (4) and generates a feature denoted **bm25**. It should be noted that, in order to differentiate the ranking algorithms and relevance features, we use different notations in Section 3 (uppercase) and here (lowercase).

In addition to these features, we also compute question relevance using two families of probability models. One is Divergence From Randomness Similarity (DFRS) [3]. The other is Information-Based Similarity (IBS) [11]. In DFRS, term weights are computed by measuring the divergence between a term distribution produced by a random process and the actual term distribution. This feature is denoted as **dfrs**. Amati et al. [3] proposed a framework for deriving probabilistic models for information retrieval. In their method, a query is assumed to be a set of independent terms whose weights are computed by measuring the divergence between a term distribution produced by a random process and the actual term distribution. Then the term weights are tuned via normalization methods to measure the similarity between document and query. Specifically, their framework builds the weighting algorithm in three steps: (i) the first step is to measure the informative content of the term in the document, and they applied seven models for this task, namely *Limiting form of Bose-Einstein*, *Geometric approximation of Bose-Einstein*, *Poisson approximation of the Binomial*, *Divergence approximation of the Binomial*, *Inverse document frequency*, *Inverse expected document frequency,* and *Inverse term frequency*; (ii) then they apply two normalization methods, *Laplace's law of succession* and *Ratio of two Bernoulli processes,* to compute the information gain when accepting the term in the document as a good document descriptor; (iii) finally, they resize the term frequency in light of the length of the document by using two hypothesis: H_1 and H_2. We further discuss the combinations of these models in the experiment (Section 5.2.2).

Similar to language models, BM25, and the DFR framework, IBS is developed based on the idea that the respective behaviors of terms in documents carry information about these words. Different from the abovementioned works, information-based frameworks use information models, and especially Shannon information is used to capture whenever a term deviates from its average behavior. As with DFR, IBS utilizes the term's informativeness to measure the query-document relevance and applies two probabilistic distributions to model term occurrence: *Log-logistic* and *Smoothed power-law*. We adopt this framework to measure the relevance between two questions.

4.3 Association Feature

The association pair refers to the pair of phrases that co-occur frequently in duplicate question pairs. They are strong indicators for identifying duplicate questions. For example, the question "*Managing several EditTexts in Android*"[5] is the duplicate of question "*android: how to elegantly set many button IDs*" in which the "*EditTexts*" and "*button IDs*" are an association pair because

[5]https://stackoverflow.com/questions/7698574/managing-several-edittexts-in-android.

"*EditTexts*" are used to set "*button IDs*". In other words, they are semantically related. Therefore, strong association between phrases in (m, t) could suggest a high similarity between these two questions. We first mine association pairs from existing duplicate question pairs. Then we generate two types of features based on the mined association pairs and denote them as *association feature* and *association score feature*. Note that we use only question titles to generate association pairs.

4.3.1 Association Pair Mining. We use existing duplicate questions (marked as "Duplicate" in Stack Overflow) and their master questions to form question pairs. For a pair of question (m, t), we learn the alignment of words between t and m via machine translation [29]. The word alignment is performed in each direction of (m, t) and (t, m) and then combined. Given the word alignments, we then extract associated phrase pairs based on 3-gram heuristics developed in Och and Ney [30] and prune those that occur less than 10 times, producing more than 130K association pairs from 25K duplicate questions. To cover a wider range of general phrase pairs, we further include a complementary set of 1.3 million association pairs [5] mined from 18 million pairs of questions from WikiAnswers.com. We denote this set of association pairs as $\mathcal{A}$; examples from $\mathcal{A}$ are ("command prompt", "console") and ("compiled executable", "exe"), and the like.

4.3.2 Association Feature Generation. Given target question t and master question m, we iterate through all spans of text $s_t \in t$ and $s_m \in m$ and check if they are associated phrases in $\mathcal{A}$. If $\langle s_t, s_m \rangle \in \mathcal{A}$, we retrieve its counts from $\mathcal{A}$ as the feature value; otherwise it is set to zero.

We also consider lexical features not only for association pairs, but also for other phrase pairs. Lexical features reflect that whether the word pairs in t and m share the same lemma, POS tag, or are linked through a *derivation* link on WordNet [5, 15]. Lexical features are only generated from the titles of m and t. For each (m, t), we iterate through spans of text and generate lexical features according to the third section of Table 1. For example, for question pair ("*Managing several EditTexts in Android*", "*android: how to elegantly set many button IDs*"), an association feature ("*EditTexts*" $\in t$, "*button IDs*" $\in m$) is generated for the span ("*EditTexts*", "*button IDs*").

5 EVALUATIONS OF DUPDETECTOR

In this section, we provide analysis on the behaviors of different features individually and their combinations. We compare our method with the state-of-the-art benchmarks and evaluate the effectiveness of the candidate selection methods.

5.1 Experimental Setup

5.1.1 Implementation. We implemented candidate selection with language models and BM25, as well as the relevance features by leveraging the lucene library[6] and developing our own Python wrapper. The parameters were set according to the suggestion in Zhai et al. [40]. For generating association feature, we used the GiZA++[7] to get word alignment-based phrase pairs and develop the association miner based on SEMPRE [4]. We used the gensim[8] implementation of `doc2vec`, and the hyper-parameter settings used in our experiments are adopted from our previous work in Zhang et al. [42]. Topic numbers in generating topical similarity ranking were set to 30 for all topic models. All experiments were conducted on a 64-bit Windows 10 Pro PC with an Intel Core i5-6440 processor and 16GB RAM.

[6] https://lucene.apache.org/.
[7] http://www.statmt.org/moses/giza/GIZA++.html.
[8] https://radimrehurek.com/gensim/.

Table 2. Statistics of Datasets

Languages	#Total	#Duplicates
Stack Overflow		
Java	716,819	28,656
C++	411,435	14,142
Python	552,695	13,823
Ruby	148,504	2,007
Html	587,499	12,080
objective-c	233,404	5,229
Quora		
NA	404,301	149,274

#Total is the total number of questions, #duplicates is the number of questions that are labeled as duplicates.

5.1.2 Datasets. We mainly used the data dump of Stack Overflow posts from April 2010 to June 2016[9] to evaluate DupDetector. The dump consists of 28,793,722 questions posted. We pruned questions without answers, producing 11,846,518 valid questions, among which 250,710 pairs of questions are marked as duplicates. Our primary dataset is the Java-related posts (identified by using tag containing "java", case-insensitive), a subset of the Stack Overflow dump. To test the robustness of DupDetector, we additionally evaluated DupDetector on subsets of other programming languages. We further used the Quora[10] dataset to evaluate the effectiveness of DupDetector on natural language questions. This dataset was released in early 2017 for the purpose of involving more research efforts in detecting duplicate posts in Quora. The statistics of evaluated datasets are detailed in Table 2. Note that the figures in the table are for valid posts. We observed from the table that the ratio of nonduplicate pairs (negative examples) to duplicate pairs (positive examples) is very skewed in Stack Overflow dataset. For example, only 0.039% of Java posts are duplicates. There have been plenty of methods in machine learning literature which deal with supervised learning for imbalanced class issues. One such popular method is to randomly sample the skewed or majority class data and make the dataset balanced [1, 13, 17, 38]. To avoid sampling bias, we drew five random samples of nonduplicate pairs, and we report our results on the average of the experimental results from the five random samples.

Taking Java posts as example, we had 28.6K Java duplicate question pairs. Accordingly, we randomly sampled 28.6K nonduplicate question pairs for each of the five random samples. Our dataset thus has 57.2K question pairs or 114.4K questions. We used all the existing duplicates to mine association pairs. To train the doc2vec and LDA models, we used all Java questions in our dataset (114.4K). We randomly sampled 5K duplicate pairs to generate association features. We used the remainder of 23.6K duplicate pairs (and the same number of nonduplicates) for evaluating DupDetector. To generate training and test partitions, we split them to a ratio of 4:1.

For other programming languages and the Quora dataset, we followed the same procedures as described for dataset generation, except for those with less than 10K duplicate pairs. For these subsets, we split them to the ratio of 1:3 for association feature generation and DupDetector evaluation, respectively. Training and test partitioning remain the same (at the ratio of 4:1).

[9] https://archive.org/details/stackexchange.
[10] https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs.

5.1.3 Preprocessing. As the Stack Overflow data dump provides texts in HTML format, we first parsed and cleaned the data, extracting the question title, body, and tags. We also used the "RelatedPostId" and "PostTypeId" information to identify duplicate questions. Then we performed a special mapping from symbols to texts tailored for programming-related questions. Specifically, we mapped all symbols used in programming languages to their symbol names (i.e., "$" to "dollar"). The set of symbols used is {!, @, #, $, %, ^, &, *, (,), (),+, {, }, {}, >>, .*, _}.[11] Note that we only map symbols that are tokenized as a single token. For example, "div[class=]" will not be converted. The mapping was performed only on question titles as question bodies may contain source code where symbol conversion might not be sensible. We then did general text processing, including stop words pruning, tokenization, and lemmatization. For the Quora dataset, we only performed general text processing as the dataset provided is clean.

5.1.4 Evaluation Metrics. We evaluated `DupDetector` using two groups of metrics. The first group of metrics evaluates the pure classification performance given various features developed. The other group of metric evaluates the overall performance of the two-stage duplication detection.

- When we analyzed the behavior of individual features, we used common classification metrics: (i) *Recall* rate reflects the ability to identify duplicate pairs among the true duplicate pairs; (ii) F_1 score is the harmonic mean of precision and recall, and is a measure of accuracy; and (iii) *Area under the Receiver Operating Characteristic (ROC) curve (AUC)* computes the probability that our model will rank a duplicate higher than a nonduplicate. A higher AUC score indicates better classification performance. The ROC curve shows how the true-positive rate changes with respect to the false-positive rate.
- Taking the ranking into consideration, the evaluation of `DupDetector` cannot solely compute the recall within the ranked questions. We also need to consider whether the duplicates are listed by the ranking algorithms. Therefore, we modified the recall as the division of the number of correctly detected duplicates to the number of total duplicates in our test dataset and refer to it as *Recall@K,* where K is the number of top-ranked questions selected for duplicate detection.

5.1.5 Classifiers. In terms of classifiers, we experimented with the following models: decision tree [7], K-Nearest Neighbors (KNN) [2], Support Vector Machines (SVM) [19], logistic regression [36], random forest [20], naive Bayes [10], and adaptive boosting (AdaBoost) [16]. For incremental (or online) learning models, we used linear SVM with Stochastic Gradient Descent (SGD) learning [41], online passive aggressive [14], and single-layer and multi-layer percepton [12].

The key parameters used in these classifiers are as follows. We set the maximum depth of the tree to 5 and used Gini impurity to measure the tree split in decision tree. We set K to 5 in KNN and weighted each neighborhood equally. In SVM, we used a linear kernel. For logistic regression, we use l_2 penalty. In random forest, the number of trees was set to 10, and the maximum depth of the tree was 5. For naive Bayes, we used a Gaussian naive Bayes classifier. In AdaBoost, the number of estimators was set to 50, while learning rate was set to 1. For incremental models, we used l_2 penalty for linear SVM with SGD, set step size as 1, and used a hinge loss function for online passive aggressive. In the multilayer perceptron, we used the α (i.e., l_2 penalty) as 1, and the number of layers was set to 50. We adopted scikit-learn[12] implementations of all the classifiers

[11] Symbols are collected from: https://www.tutorialspoint.com/computer_programming/computer_programming_characters.htm.
[12] http://scikit-learn.org/stable/.

Table 3. Performance of Different Combinations of Vector Similarity Features in DupDetector and VSM

Classifiers	title	body	tags	title+body	title+body+ tags	title+body, title, body	all*	VSM
Recall								
KNN	0.540	0.635	0.538	0.66	0.715	0.789	**0.830**	0.732
Linear SVM	0.440	0.689	0.42	0.732	0.742	0.726	**0.787**	0.608
RBF SVM	0.420	0.617	0.418	0.669	0.696	0.788	**0.887**$^{\triangle}$	0.686
Naive Bayes	0.409	0.694	0.408	0.737	0.754	707	**0.768**	0.678
Logistic Regression	0.484	0.686	0.476	0.736	0.756	0.767	**0.855**	0.736
Decision Tree	0.506	0.635	0.371	0.651	0.677	0.757	**0.833**	0.760
Random Forest	0.474	0.603	0.419	0.664	0.692	0.739	**0.803**	0.722
AdaBoost	0.442	0.566	0.415	0.654	0.687	0.753	**0.858**	0.696
F_1								
KNN	0.556	0.666	0.545	0.707	0.754	0.823	**0.863**	0.780
Linear SVM	0.508	0.708	0.501	0.758	0.784	0.752	**0.815**	0.743
RBF SVM	0.419	0.688	0.499	0.741	0.770	0.827	**0.897**$^{\triangle}$	0.788
Naive Bayes	0.328	0.708	0.493	0.759	0.784	0.753	**0.803**	0.785
Logistic Regression	0.467	0.706	0.529	0.758	0.785	0.795	**0.871**	0.803
Decision Tree	0.572	0.691	0.474	0.733	0.760	0.796	**0.849**	0.811
Random Forest	0.558	0.691	0.529	0.743	0.767	0.810	**0.838**	0.798
AdaBoost	0.540	0.658	0.507	0.736	0.767	0.799	**0.881**	0.790

*__all__ is five-dimensional combining title, body, tag, title+body and title+body+tag vectors; first five feature columns are one-dimensional; the sixth column (**title+body, title, body**) is three-dimensional.
Bold values indicate the best result among all features with a certain classifier. **Bold**$^{\triangle}$ refers to the best result among all classifiers.

just mentioned. All other parameters were set using the default settings in scikit-learn. For each classifier, we reported the average results of five trials.

5.2 Results

In this section, we first report the behaviors of features in terms of recall and F_1 score within each category of features (Section 5.2.1 to Section 5.2.3). Next, we analyze classification performance using three categories of features independently and various combinations of them (Section 5.2.4). These evaluations are performed on Java-related questions. Then we experiment DupDetector with several classifiers and compare it to state-of-the-art systems on a range of programming languages. (Section 5.2.5). Finally, runtime analysis is reported for different ranking algorithms in DupDetector and a detection method without ranking (Section 5.2.6).

5.2.1 Vector Similarity Feature Analysis. As described in Section 4.1, we trained five doc2vec models, hence we obtained 10 vectors for each question pair. Cosine similarities on the five pairs of vectors were computed, and we denote (for example) title vector similarity as **title** in Table 3, where the classification performances of multiple combinations of these five vector similarities using a range of classifiers are detailed. The first five feature columns are one-dimensional features; namely, for each question pair, only one value is used for classification. The next two columns are combinations of vector similarities: **title+body, title, body** is three-dimensional, and **all** is five-dimensional. The last column contains the baseline tf-idf document vectors (VSM).

The results show that among one-dimensional features, vectors computed on longer texts performs better than shorter texts: **tag** produces the lowest recall, and **title** is slightly better than **tags,** following **body** and **title+body**, which is the vector learned on the concatenation of title and body texts. **title+body+tag** is learned on the concatenation of title, body, and tags texts and gives the best among the five, but is still worse than VSM, which produces 73.2% recall. Three-dimensional feature **title+body, title, body** produces better recall than one-dimensional features and VSM, giving 78.9% recall. When considering all the vectors, the five-dimensional feature **all** gives the highest recall and achieves more than 75% recall for all the classifiers evaluated, among which RBF SVM produces 88.7% recall. In terms of F_1 score, the results are consistent with recall: Higher dimensional features perform better than lower dimensional features; **all** achieves the highest F_1 score for all the evaluated classifiers compared to other features. The best F_1 score is achieved using the RBF SVM classifier (0.897). Therefore, we chose **all**, the five-dimensional feature, to represent the vector similarity feature in the following experiments.

The classifiers performed differently on features with different dimensions. For the five-dimensional feature, RBF SVM produces the best recall, followed by AdaBoost and Logistic Regression, then Decision tree and KNN. Linear SVM and Naive Bayes are worse than the other classifiers. These results indicate that RBF SVM is the best classifier for the vector similarity feature.

It is important to note that the dimension of vectors learned by doc2vec will affect the effectiveness of the vector similarity feature. We observed in extensive analysis that higher dimension vectors produce better classification performance. We only report the results based on the 100-dimensional (see Zhang et al. [42] for doc2vec parameter settings) doc2vec vectors in this article.

5.2.2 Relevance Feature Analysis. The relevance feature considers multiple probability models to produce relevance scores between questions. We denote the language model using Jelinek-Mercer smoothing as $\mathbf{lm_{jm}}$ and using Bayesian smoothing with Dirichlet priors as $\mathbf{lm_{dir}}$. As described in Section 4.2, DFR provides multiple models. It supports seven basic models and two normalization methods, producing 14 combinations. Due to the limited space, we only chose the combinations that performed well in Amati and Van Rijsbergen [3], which are B_eB_2, GB_2, and $I(F)B_2$, in which B_e is the *Limiting form of Bose-Einstein*, G refers to *Geometric approximation of Bose-Einstein,* and $I(F)$ represents Inverse term frequency. We denote these three solutions as $\mathbf{dfrs\text{-}B_eB_2}$, $\mathbf{dfrs\text{-}GB_2}$ and $\mathbf{dfrs\text{-}I(F)B_2}$ respectively. In the IB framework, two probability models are provided, and we denote the two solutions as $\mathbf{ibs\text{-}SPL}$ for smoothed power-law and $\mathbf{ibs\text{-}LL}$ for log-logistic. For the relevance feature computed via BM25, we denote this feature as **bm25**.

From Table 4, we observe that **bm25** gives the best recall among the five one-dimensional features on most of the classifiers except for Linear SVM, with which $\mathbf{lm_{jm}}$ performs the best (75.1%), and Random Forest, with which $\mathbf{ibs\text{-}LL}$ produces the highest recall (74.1%). The two features computed from IB perform slightly differently, and, in most cases, $\mathbf{ibs\text{-}LL}$ gives better results compared to $\mathbf{ibs\text{-}SPL}$. The three features derived from DFR show similar performance with $\mathbf{dfrs\text{-}I(F)B_2}$ which achieves the best recall for most classifiers. From the results, we can see DBRS and IB features interchangeably give better recall across all the classifiers evaluated, indicating they are similar. For query likelihood-based features $\mathbf{lm_{jm}}$ and $\mathbf{lm_{dir}}$, $\mathbf{lm_{jm}}$ gives better performance than $\mathbf{lm_{dir}}$ in terms of recall with most of the classifiers, except for Random Forest. It is hard to tell which feature is better than another as the recall varies with different classifiers. The eight-dimensional feature **all**, the combination of the eight features, always gives the highest recall. Similarly, **bm25** produces the highest F_1 score for most classifiers compared to other one-dimensional features, and **all** always performs the best in terms of F_1 score, with the highest value (0.842) when using AdaBoost.

Table 4. Performance of Different Combinations of Relevance Features in DupDetector

Classifiers	lm_{jm}	lm_{dir}	ibs-LL	ibs-SPL	dfrs-B_eB_2	dfrs-GB_2	dfrs-$I(F)B_2$	bm25	all*
Recall									
KNN	0.903	0.898	0.908	0.901	0.908	0.909	0.912	0.913	**0.917**△
Linear SVM	0.751	0.677	0.722	0.717	0.711	0.713	0.717	0.742	**0.759**
RBF SVM	0.750	0.735	0.735	0.731	0.742	0.741	0.749	0.756	**0.760**
Naive Bayes	0.859	0.692	0.859	0.853	0.854	0.855	0.859	0.859	**0.863**
Logistic Regression	0.638	0.551	0.673	0.653	0.657	0.658	0.667	0.664	**0.697**
Decision Tree	0.725	0.705	0.735	0.733	0.740	0.743	0.749	0.750	**0.756**
Random Forest	0.719	0.732	0.741	0.729	0.726	0.725	0.731	0.724	**0.755**
AdaBoost	0.748	0.746	0.756	0.716	0.746	0.745	0.745	0.760	**0.763**
F_1									
KNN	0.675	0.691	0.683	0.680	0.677	0.678	0.683	0.685	**0.693**
Linear SVM	0.814	0.797	0.806	0.800	0.799	0.801	0.805	0.830	**0.834**
RBF SVM	0.810	0.829	0.807	0.799	0.813	0.811	0.821	0.830	**0.836**
Naive Bayes	0.774	0.725	0.774	0.769	0.766	0.767	0.774	0.774	**0.781**
Logistic Regression	0.760	0.706	0.788	0.768	0.776	0.776	0.783	0.784	**0.809**
Decision Tree	0.797	0.813	0.807	0.803	0.811	0.814	0.821	0.829	**0.837**
Random Forest	0.801	0.828	0.822	0.811	0.804	0.807	0.813	0.824	**0.835**
AdaBoost	0.814	0.829	0.828	0.821	0.814	0.815	0.817	0.835	**0.842**△

*__all__ is eight-dimensional combining all features.
Bold values indicate the best result among all features with a certain classifier. **Bold**△ refers to the best result among all classifiers.

Interestingly, KNN produces the highest recall (91.5%) for the **all** feature. However, KNN gives the lowest F_1 score for **all**, indicating that the precision given by KNN on **all** is very low. Naive Bayes produces the second best recall but gives a lower F_1 score compared to AdaBoost. As a balance between recall and F_1 score, we considered Naive Bayes and AdaBoost to be the most suitable classifiers for the relevance feature and used **all** to represent the relevance feature in the following evaluations.

5.2.3 Association Feature Analysis. As the association feature is a sparse feature in a very high-dimensional space, we additionally chose some online learning models due to the computation requirement. Table 5 details the classification performance of the association feature on a range of classifiers. KNN and SVMs fail on the association feature, showing they are not scalable. Among other offline classifiers, Logistic Regression (86.1%) outperforms others, followed by AdaBoost. For online learning algorithms, multi-layer perceptron produces the best recall (86.8%). Linear SVM with SGD is slightly worse in terms of recall (86.3%). Similarly, for F_1 score, Logistic Regression gives the highest value (0.872) among offline classifiers, while multi-layer perceptron is the best in online algorithms, achieving a 0.866 F_1 score. The results show that online learning is not necessarily better than offline learning algorithms, but the best recall is achieved by the online learning algorithm. Therefore, we considered multi-layer perceptron as the best classifier for the association feature.

When we mined the association pairs, we used all the duplicate questions in our dataset, regardless of the differences among programming languages. Intuitively, mining association pairs separately for language-specific questions would result in a more accurate association feature, producing better classification results. We have not addressed this issue in this work and will leave it for the future.

Table 5. Performance of Association Features in DupDetector

Classifiers	Recall	F_1
KNN (K=5)	-*	-
Linear SVM	-	-
RBF SVM	-	-
Naive Bayes	0.831	0.826
Logistic Regression	**0.861**	**0.872**
Decision Tree	0.816	0.817
Random Forest	0.752	0.666
AdaBoost	0.845	0.854
Linear SVM with SGD	0.863	0.828
Aggregate SGD	0.850	0.855
On-line passive aggressive	0.785	0.833
Single-layer Perceptron	0.837	0.846
Multi-layer Perceptron	**0.868**	**0.866**

*'-' denotes corresponding algorithm fails on association feature.
Bold values indicate the best result among one of the two groups of classifiers.

5.2.4 Performance Comparison of the Three Categories of Features. In previous sections, we reported our analysis of DupDetector's classification performance using the following features independently: vector similarity (VS), relevance (RE), and association (AS). In this evaluation, we compared the performance of these features' various combinations. As reported in Section 5.2.1, RBF SVM produces the best performance for VS. However, it fails on AS. Thus we did not choose it as classifier here. AdaBoost and naive Bayes are reported the most suitable for RE in Section 5.2.2, and AdaBoost is better for VS and ASS. Although logistic regression gives the worst performance for RE, it shows effectiveness for VS and AS. Muli-layer perceptron is the best for AS, as shown in Section 5.2.3. Therefore, we compared the feature combinations over naive Bayes, AdaBoost, logistic regression, and multi-layer perceptron and reported the results in Table 6.

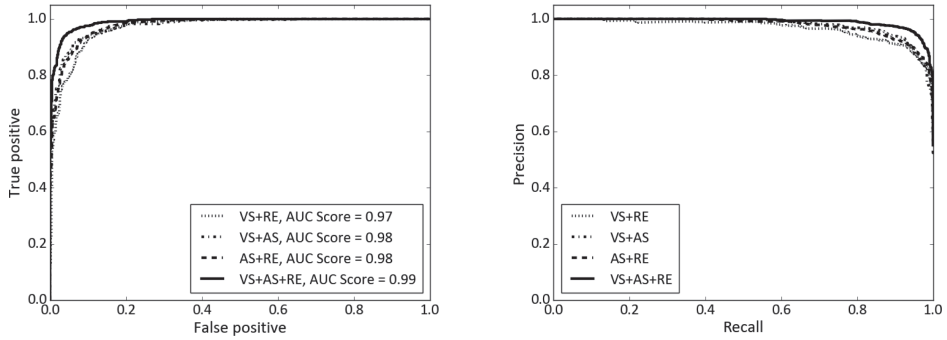
In Table 6, classification recall and F_1 score using different combinations of features over the selected four classifiers are summarized. When working independently, VS, RE, and AS perform the best interchangeably with different classifiers. For example, when using Naive Bayes, RE is the best feature. But for AdaBoost, VS gives the highest recall and F_1 score. The combination of any two of the features outperforms a single feature, suggesting that these three features complement each other. When combining all the individual features (VS+AS+RE), the best performance of a certain classifier is always achieved. For example, Naive Bayes achieves more than 0.890 in terms of recall and 0.841 for the F_1 score. Multi-layer perceptron always gives the best performance among the four classifiers with different feature combinations. For VS+AS+RE, it achieves 96.3% recall and a 0.954 F_1 score. Therefore, we used multi-layer perceptron as the classifier and VS+AS+RE as the feature in DupDetector for the rest evaluations.

Figure 3 presents the ROC curve and AUC score when these features were used in combinations. The results are consistent with the recall and F_1 score: VS+AS+RE always give the best performance.

Table 6. Performance of Different Combinations of Three Categories of Features in DupDetector

Classifiers	VS	RE	AS	VS + RE	VS + AS	AS + RE	VS + AS + RE
Recall							
Naive Bayes	0.768	0.863	0.831	0.862	0.832	0.863	**0.890**
AdaBoost	0.858	0.763	0.845	0.921	0.921	0.936	**0.955**
Logistic Regression	0.855	0.697	0.861	0.946	0.919	0.948	**0.963**
Multi-layer Perceptron	0.836	0.924	0.868	0.946	0.931	0.953	**0.963**$^{\triangle}$
F_1							
Naive Bayes	0.803	0.781	0.826	0.821	0.827	0.832	**0.841**
AdaBoost	0.881	0.842	0.854	0.926	0.931	0.915	**0.939**
Logistic Regression	0.871	0.809	0.872	0.927	0.930	0.927	**0.949**
Multi-layer Perceptron	0.852	0.867	0.866	0.915	0.925	0.924	**0.954**$^{\triangle}$

Bold values indicate the best result among all features with a certain classifier. **Bold**$^{\triangle}$ refers to the best result among all classifiers.



(a) ROC Curve (b) Precision-Recall Curve

Fig. 3. ROC and precision-recall curve on different combination of features in DupDetector.

5.2.5 Performance Comparison to the State of the Art. We compared DupDetector to state-of-the-art Dupe [1] on PCQA duplicate detection over seven programming languages using different ranking methods. As is Dupe, DupDetector is a two-stage "ranking-classification" method. Thus, we used the defined *Recall@K* metric to evaluate their performance, and this recall differs from the recall we used in previous evaluations. In this section, we refer to *Recall@K* as recall for simplicity. In order to compare, we reimplemented Dupe according to the described heuristics in Ahasanuzzaman et al. [1] because both the data and implementation of the techniques are not available. We can not guarantee that our implementation is errorless, but we tried our best to capture all the details of Dupe. Interestingly, the performance values are slightly better than the reported values in Ahasanuzzaman et al. [1]. We only report the suggested number of K with the best performance of Dupe due to limited space.

Table 7 presents the results over classification recall. Dupe adopts BM25 as the ranking method and uses Random Forest as the classifier. For DupDetector, we examined the impact of four ranking methods to the recall. Note that we used different notations for ranking methods (uppercase) and relevance features (lowercase). Topical ranking gives the worst performance, suggesting that the topical similarity is inaccurate when examining the similarity between questions. The two query likelihood methods produce similar recall across the seven languages: LMJM outperforms

Table 7. *Recall@K* on Dupe and DupDetector with Different Ranking Methods

	Dupe@20	DupDetector@20			
Languages	**BM25**	**Topical**	**LMJM**	**LMDIR**	**BM25**
Java	0.537	0.505	0.774	0.772	**0.782**
C++	0.516	0.492	0.772	0.769	**0.779**
Python	0.542	0.501	0.776	0.779	**0.786**
Ruby	0.668	0.577	0.696	0.701	**0.705**
Html	0.509	0.482	0.733	0.735	**0.736**
Objective-C	0.564	0.513	0.728	0.727	**0.733**

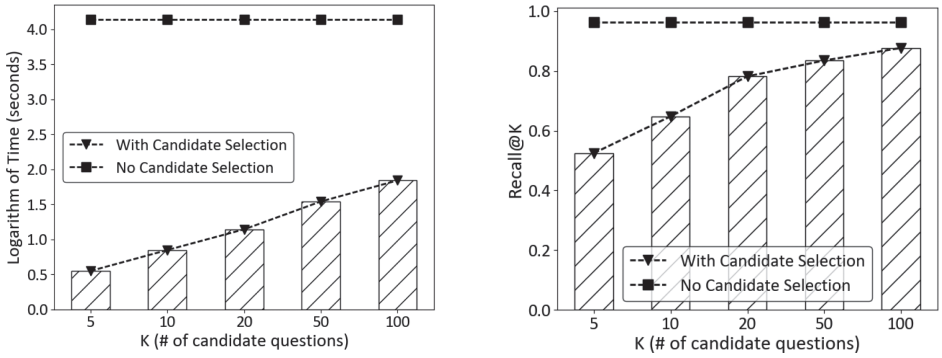
Table 8. Average Feature Generation Time for a Question Pair in DupDetector

	VS	**AS**	**RE**
Time (sec.)	0.635	0.061	6.575e-5

LMDIR in Java, C++, and Objective-C questions but is worse in other languages. They are both worse than the BM25 ranking. Then we compared DupDetector with BM25 to Dupe. For all seven languages, DupDetector outperforms Dupe by at least 4% (Ruby). The biggest improvement is on C++, where the gain is greater than 25%. The significant performance gap highlights the strength of DupDetector.

5.2.6 Runtime Analysis. To evaluate the effectiveness of candidate selection, we analyzed the time used for duplicate detection with and without candidate selection. We used the Java questions in this evaluation. We excluded the time used to train the doc2vec model and classifiers and mine association pairs as they are common actions and can be done separately from the detection process. Therefore, the time for ranking-based method consists of ranking, pairing, and feature generation time. On the other hand, the time used for methods without candidate selection only contains the pairing and feature generation time but needs to consider all historical questions. Ranking time depends on the number of total historical questions, and pairing time depends on how many questions need to be examined given a newly issued question. Table 8 lists the average feature generation time for a question pair in our implemented DupDetector. Interestingly, VS computation occupies the largest part of the time, which is 0.635 seconds. A possible reason is that the model trained on a large corpus of questions, leading to time-consuming vector generation. RE generation takes up the least time, 6.575e-5 seconds, while the generation of AS takes 0.061 seconds for each question pair in average.

Furthermore, we illustrated the comparison of the average time used for the evaluated methods on different numbers of candidate questions in Figure 4(a). We used a logarithm plot (base=10) to clearly show the performance difference. The larger gap between the two lines showcases that the method with ranking stage considerably reduces computation cost and hence accelerates the duplicate detection process. Moreover, consistent with common cognition, when K increases, the time used increases. To further clearly showcase the impact of candidate number K, we also depict the overall *Recall@K* given different K when applying or not applying candidate selection in Figure 4(b). The larger K is, the better recall it achieves. However, performance still cannot compete with the method without applying candidate selection. This result is consistent with our previous evaluation, where *Recall@K* in Table 7 has a lower value than recall in Table 6. The reason is that some possible duplicate questions could be mistakenly ranked very low in



(a) Average Runtime (b) Recall@K

Fig. 4. Performance comparison on methods with/without candidate selection.

Table 9. Performances of DupDetector on the Quora Dataset. K=20

Recall@20	F_1	AUC Score	Average Time
0.815	0.823	0.994	10.477 sec.

the ranking stage and are not included in the top K candidates to be examined. To this end, in practice, it is necessary to compromise between efficiency and accuracy.

5.2.7 Evaluation on CQA Dataset. In this section, we report the evaluation of DupDetector on the Quora dataset. The difference is that the Quora dataset contains questions using normal, natural languages, unlike the Stack Overflow dataset that contains symbols that are typically not seen in natural languages. The intent of this evaluation is to examine the effectiveness of DupDetector on more general CQA duplicate detection. Table 9 reports the results of four metrics when K is set to 20. *Recall*@20 achieves 81.5%, which is slightly higher than the results reported in Table 7. The reason is that the programming-related questions might cause some noise because the meanings of symbols are difficult to comprehend automatically. Similarly, the AUC score is higher than results produced by programming language-related questions. The average duplicate detection time given a question is 10.477 seconds, which is shorter than the time used by Stack Overflow questions due to their simpler preprocessing step. The results showcase that DupDetector is also effective at detecting duplicates for normal CQA platforms, indicating that it is a generic approach.

6 RELATED WORK

Our work belongs to the task of *mining CQA platforms* and is to some extent related to *paraphrase recognition* in the broader NLP community.

Mining CQA Platforms. Extensive research efforts have been devoted to mining CQA platforms due to the increasing popularity of CQA. Most works focused on answer retrieval, where the basic idea is to answer new question with the answers to similar historical questions or that are semantically similar to the new question. Cao et al. [9] explored category information in Yahoo! Answers and combined a language model with a translation model to estimate the relevance of existing question–answer pairs to a query question. Wang et al. [37] identified similar questions by assessing the similarity of their syntactic parse trees. Yang et al. [38] classified questions based on heuristics and topical information to predict whether they will be answered.

The work also empirically proved that classification-based methods outperform word-based methods. Zhou et al. [44] studied synonymy and polysemy to measure question similarity and built a concept thesaurus from Wikipedia for the task, with the ultimate goal of improving question-answering from Yahoo! Answers. In recent years, programming CQA has been gaining significant momentum and attracting increasing interest from the community. Treude et al. [35] analyzed the types of questions posted on Stack Overflow and examined which kind of questions were well answered and which remained unanswered. Correa et al. [13] studied the characteristics of deleted questions on Stack Overflow to predict the possibility of deletion of a newly issued question. Zhang et al. [43] proposed the first work to leverage supervised learning to detect duplicate question. They used textual features derived from title, description, topic and, tag to build the classifiers. Ahasanuzzaman et al. [1] conducted an extensive analysis to understand why duplicate questions are created on Stack Overflow, and adopted the binary classification idea to identify duplicate questions. Very recently, Zhang et al. [42] extracted features for Stack Overflow question pairs by leveraging latent semantics learned through `word2vec` and frequently co-occurring phrase pairs mined from existing duplicate question pairs. These features produce strong performance for duplicate detection from Stack Overflow posts.

Paraphrase recognition. Paraphrase recognition (or paraphrase detection, paraphrase identification) aims to examine whether two texts convey exactly the same information or have similar implications. Extensive research efforts have been devoted to this task [18, 22, 26, 27, 31, 34]. Mihalcea et al. [27] measured text similarities by the similarities of component words, which are computed using corpus-based and knowledge-based word semantic similarity measures. This work is considered as the baseline for many following research efforts. Qiu et al. [31] detected dissimilarities between phrases and made judgments based on the significance of such dissimilarities. This method showed a key benefit of explaining the cause of the nonparaphrase sentence pair. Socher et al. [34] learned feature vectors for phrases in syntactic trees and introduced dynamic pooling to compute fixed-sized vectors from variable-sized matrices. The pooled representation is then used as input to a classifier. Madnani et al. [26] performed classification on the features extracted from the combination of eight machine translation metrics. Ji et al. [22] utilized latent representation from matrix factorization as features to perform supervised learning on phrases. He et al. [18] employed a Convolutional Neural Networks (CNN) model to learn sentence representations at multiple levels of granularity. Then the information of two sentences is combined by matching multiple distance heuristics. Paraphrase recognition techniques were recently adopted in the QA communities to examine the quality of question paraphrase [39], which is the rewriting of the original question, with the ultimate goal of improving the recall of QA.

7 CONCLUSION AND FUTURE WORK

We introduced `DupDetector`, a new state-of-the-art duplicate detection system for the PCQA domain. `DupDetector` is a two-stage "ranking-classification" that efficiently and accurately identifies duplicate questions from a large amount of historical questions. `DupDetector` consists of a ranking-based candidate selection stage to reduce the search space for possible duplicates and a classification stage driven by a few features derived using methods from the deep learning and information retrieval literature. Leveraging prominent ranking algorithms and combining all features in a classification model, `DupDetector` outperforms state-of-the-art duplicate detection systems by at least 4%, and, in some cases, more than 25% in multiple programming languages in terms of recall rate. As a product of the association feature, we have mined a set of associated phrases from duplicate questions on Stack Overflow. These phrases are domain-specific to PCQA and could be used in other tasks such as keyword recommendation for forum searching.

Despite the strong performance of DupDetector, there is room for improvement. For example, DupDetector does not identify that "*isset in Jquery?*" is a duplicate of "*Finding whether the element exists in whole html page*". After analysis, we find that the first question asks for functions in JQuery that are similar to "isset" in PHP, which crosses the programming language boundary. We will leave the development of techniques that tackle cross-language duplicate detection for future work. Further, we can extract more features from different aspects (e.g., the users' profiles). This information could suggest the newly issued question's quality by considering the reputation score and expertise level of the system.

REFERENCES

[1] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. Mining duplicate questions in stack overflow. In *Proceedings of of the MSR 2016*. ACM, Austin, Texas, USA, 402–412.

[2] Naomi S. Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician* 46, 3 (1992), 175–185.

[3] Gianni Amati and Cornelis Joost Van Rijsbergen. 2002. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems* 20, 4 (2002), 357–389.

[4] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the EMNLP 2013*. ACL, Seattle, Washington, USA, 1533–1544.

[5] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proceedings of the ACL 2014*. 1415–1425.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.

[7] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.

[8] Xin Cao, Gao Cong, Bin Cui, and Christian S. Jensen. A generalized framework of exploring category information for question retrieval in community question answer archives. In *Proceedings of the WWW 2010*. ACM, Raleigh, North Carolina, USA, 201–210.

[9] Xin Cao, Gao Cong, Bin Cui, Christian S. Jensen, and Quan Yuan. 2012. Approaches to exploring category information for question retrieval in community question-answer archives. *ACM Transactions on Information Systems* 30, 2 (2012), 7.

[10] Tony F. Chan, Gene Howard Golub, and Randall J. LeVeque. Updating formulae and a pairwise algorithm for computing sample variances. In *Proceedings of the COMPSTAT 1982*. Springer, Physica, Heidelberg, 30–41.

[11] Stéphane Clinchant and Éric Gaussier. Information-based models for ad hoc IR. In *Proceedings of the SIGIR 2010*. ACM, Geneva, Switzerland, 234–241.

[12] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the EMNLP 2002*. ACL, Philadelphia, PA, USA, 1–8.

[13] Denzil Correa and Ashish Sureka. Chaff from the wheat: Characterization and modeling of deleted questions on stack overflow. In *Proceedings of the WWW 2014*. ACM, Seoul, Republic of Korea, 631–642.

[14] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research* 7 (2006), 551–585.

[15] C. Fellbaum. 1998. WordNet: An electronic lexical database. MIT Press.

[16] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the EuroCOLT 1995*. Springer, Barcelona, Spain, 23–37.

[17] Haibo He and Edwardo A. Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering* 21, 9 (2009), 1263–1284.

[18] Hua He, Kevin Gimpel, and Jimmy J. Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the EMNLP 2015*. ACL, Lisbon, Portugal, 1576–1586.

[19] Marti A. Hearst, Susan T. Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. 1998. Support vector machines. *IEEE Intelligent Systems and Their Applications* 13, 4 (1998), 18–28.

[20] Tin Kam Ho. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 8 (1998), 832–844.

[21] Fred Jelinek and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the PRNI 1980*. North Holland, Amsterdam, Netherlands, 381–397.

[22] Yangfeng Ji and Jacob Eisenstein. Discriminative improvements to distributional sentence similarity. In *Proceedings of the EMNLP 2013*. ACL, Seattle, Washington, USA, 891–896.

[23] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Proceedings of the RepL4NLP 2016*. ACL, Berlin, Germany, 78–86.

[24] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the ICML 2014*. JMLR.org ©2014, Beijing, China, 1188–1196.
[25] Chenliang Li, Haoran Wang, Zhiqian Zhang, Aixin Sun, and Zongyang Ma. Topic modeling for short texts with auxiliary word embeddings. In *Proceedings of the SIGIR 2016*. ACM, Pisa, Italy, 165–174.
[26] Nitin Madnani, Joel R. Tetreault, and Martin Chodorow. Re-examining machine translation metrics for paraphrase identification. In *Proceedings of the NAACL 2012*. ACL, Montréal, Canada, 182–190.
[27] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the AAAI 2006*. AAAI Press, Boston, Massachusetts, USA, 775–780.
[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the NIPS 2013*. Neural Information Processing Systems, Lake Tahoe, Nevada, USA, 3111–3119.
[29] Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics* 29, 1 (2003), 19–51.
[30] Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics* 30, 4 (2004), 417–449.
[31] Long Qiu, Min-Yen Kan, and Tat-Seng Chua. Paraphrase recognition via dissimilarity significance classification. In *Proceedings of the EMNLP 2006*. ACL, Sydney, Australia, 18–26.
[32] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline M. Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In *Proceedings of the TREC 1994*. National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, USA, 109–126.
[33] Anna Shtok, Gideon Dror, Yoelle Maarek, and Idan Szpektor. Learning from the past: Answering new questions with past answers. In *Proceedings of the WWW 2012*. ACM, Lyon, France, 759–768.
[34] Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proceedings of the NIPS 2011*. Neural Information Processing Systems, Lake Tahoe, Nevada, United States, 801–809.
[35] Christoph Treude, Ohad Barzilay, and Margaret-Anne D. Storey. How do programmers ask and answer questions on the web? In *Proceedings of the ICSE 2011*. ACM, Waikiki, Honolulu, HI, USA, 804–807.
[36] Strother H. Walker and David B. Duncan. 1967. Estimation of the probability of an event as a function of several independent variables. *Biometrika* 54, 1–2 (1967), 167–179.
[37] Kai Wang, Zhaoyan Ming, and Tat-Seng Chua. A syntactic tree matching approach to finding similar questions in community-based QA services. In *Proceedings of the SIGIR 2009*. ACM, Boston, MA, USA, 187–194.
[38] Lichun Yang, Shenghua Bao, Qingliang Lin, Xian Wu, Dingyi Han, Zhong Su, and Yong Yu. Analyzing and predicting not-answered questions in community-based question answering services. In *Proceedings of the AAAI 2011*. AAAI Press, San Francisco, California, USA, 1273–1278.
[39] Pengcheng Yin, Nan Duan, Ben Kao, Jun-Wei Bao, and Ming Zhou. Answering questions with complex semantic constraints on open knowledge bases. In *Proceedings of the CIKM 2015*. ACM, Melbourne, Australia, 1301–1310.
[40] Cheng Xiang Zhai and John D. Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems* 22, 2 (2004), 179–214.
[41] Tong Zhang. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the ICML 2004*. ACM, Banff, Alberta, Canada, 919–926.
[42] Wei Emma Zhang, Quan Z. Sheng, Jey Han Lau, and Ermyas Abebe. Detecting duplicate posts in programming QA communities via latent semantics and association rules. In *Proceedings of the WWW 2017*. ACM, Perth, Australia, 1221–1229.
[43] Yun Zhang, David Lo, Xin Xia, and Jianling Sun. 2015. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology* 30, 5 (2015), 981–997.
[44] Guangyou Zhou, Yang Liu, Fang Liu, Daojian Zeng, and Jun Zhao. Improving question retrieval in community question answering using world knowledge. In *Proceedings of the IJCAI 2013*. IJCAI/AAAI, Beijing, China, 2239–2245.

Received June 2017; revised October 2017; accepted November 2017