Detecting Duplicate Posts in Programming QA Communities via Latent Semantics and Association Rules

Wei Emma Zhang[†], Quan Z. Sheng[‡], Jey Han Lau[*◇], Ermyas Abebe[◇]

[†]School of Computer Science, The University of Adelaide, Adelaide, Australia
[‡]Dept of Computing, Macquarie University, Sydney, Australia
[*]Dept of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
[◇]IBM Research Australia, Melbourne, Australia

wei.zhang01@adelaide.edu.au, michael.sheng@mq.edu.au,
jeyhan.lau@gmail.com, etabebe@au1.ibm.com

ABSTRACT

Programming community-based question-answering (PCQA) websites such as Stack Overflow enable programmers to find working solutions to their questions. Despite detailed posting guidelines, duplicate questions that have been answered are frequently created. To tackle this problem, Stack Overflow provides a mechanism for reputable users to manually mark duplicate questions. This is a laborious effort, and leads to many duplicate questions remain undetected. Existing duplicate detection methodologies from traditional community based question-answering (CQA) websites are difficult to be adopted directly to PCQA, as PCQA posts often contain source code which is linguistically very different from natural languages. In this paper, we propose a methodology designed for the PCQA domain to detect duplicate questions. We model the detection as a classification problem over question pairs. To extract features for question pairs, our methodology leverages continuous word vectors from the deep learning literature, topic model features and phrases pairs that co-occur frequently in duplicate questions mined using machine translation systems. These features capture semantic similarities between questions and produce a strong performance for duplicate detection. Experiments on a range of real-world datasets demonstrate that our method works very well; in some cases over 30% improvement compared to state-of-the-art benchmarks. As a product of one of the proposed features, the association score feature, we have mined a set of associated phrases from duplicate questions on Stack Overflow and open the dataset to the public.

Keywords

Community-based question answering; Latent semantics; Association rules; Question quality; Classification

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052701

1. INTRODUCTION

Community-based question answering (CQA) sites such as Quora[1], Baidu Zhidao[2], and Stack Exchange[3] have grown in popularity in the recent years. CQA sites are a promising alternative to traditional web search as users have queries that are often subjective, open-ended and require expert opinions. To cater the multitude of interests for its community, Stack Exchange has a set of sub-domains that focus on a particular subject or topic.

Stack Overflow, a programming community question answering site (PCQA), is a sub-domain in Stack Exchange created for programming related questions. Despite detailed guidelines on posting ethics (and possibly the users' best intentions), a large number of created questions are poor in quality [11]. 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. This approach is laborious, but more importantly, a large number of duplicate questions remain undetected. A high quality duplication detection system will considerably improve user experience: for inexperienced users creating a new question, it can suggest a related post before posting; for experienced users it can suggest potential duplicate posts for manual verification.

Question duplication is a pervasive issue in CQA in general, and a number of studies have looked into related problems, including finding similar questions [27, 32, 6, 7], and generating answers for new questions from past answers [23, 28]. These works framed the task as a classification or prediction task, and relied on a number of extracted features to train a model. It is important to note, however, that features explored in these methods may not necessarily suitable to PCQA as PCQA posts often contain source code from programming languages which are linguistically very different to natural languages.

There are few studies that explored question duplication for the PCQA domain [1, 31]. The work in [31] tackled duplicate question detection on Stack Overflow by generating features using title, description, topical and tag information

[1]https://www.quora.com/
[2]http://zhidao.baidu.com/
[3]http://stackexchange.com/

to classify whether a question pair is a duplicate. The work in [1] improved this method by extending features adopted from [24] which mined information from Twitter posts. Additionally, they conducted extensive analysis to learn why users create duplicate posts. In this paper, we seek to improve upon the benchmark performance set by their system.

Our methodology 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 `PCQADup`. Given a question pair, `PCQADup` generates three types of features. *Vector similarity*, the first of its features, represents questions as continuous vectors in a high-dimensional space. In the deep learning literature, `word2vec` [20] was proposed to learn word vectors/embeddings efficiently from large text corpora, and it has been shown to effectively capture semantics of words. `doc2vec`, an extension of `word2vec`, is developed to generate embeddings for any arbitrary word sequences [18]. Inspired by the success of these neural methods, we compute the `doc2vec` representation of the title and body content of a post, and measure similarity between a question pair based on vector cosine similarity measures. The second type of features is *topical similarity*, computed using a topic model for extracting themes from short texts. The similarity of a question pair is measured by computing the similarity of topical distributions between the pair. The last type of features is *association score*. We first mine association pairs, i.e., pairs of phrases that co-occur frequently in known duplicate questions, by adopting a word alignment method developed in the machine translation literature. To generate the association score for a question pair, we train a perceptron that takes association pairs and lexical features as input. The idea of using associated phrases has been explored in knowledge base question answering (KBQA) for ranking generated queries in curated KBQA [4] or measuring the quality of question paraphrases in open KBQA [29]. Our work is the first to adapt the idea to detect duplicate questions in PCQA websites.

To summarise, the main contributions of the paper are:

1) **Novel features for duplicate question detection**: We represent questions as continuous vectors in a high dimensional space. We show that neural embeddings capture semantic similarity better compared to traditional vector space representation such as `tf-idf` in the task of duplicate question detection.

2) **Association pairs for PCQA**: We mine over 130K association pairs from known duplicate questions in Stack Overflow. The data contains phrases pairs that frequently occur in duplicate question pairs and are domain-specific to PCQA. The source code for mining association pairs and the mined pairs are publicly available for download[4].

3) **Extensive experimental evaluation**: In addition to Stack Overflow, we also test `PCQADup` on other Stack Exchange sub-domains. Results suggest that `PCQADup` outperforms state-of-the-art benchmark by over 30% in terms of recall for duplicate detection.

The rest of paper is organised as follows. In Section 2, we give a high level perspective of our system `PCQADup`. In Section 3, we describe features we have developed for `PCQADup`. We report experimental results in Section 4. In Section 5, we review related work. Finally, we discuss implications, caveats and conclude in Section 6.

[4]`http://weiemmazhang.me/codes/pcqadup.html`

2. METHODOLOGY OVERVIEW

`PCQADup` has three components: 1) pre-processing; 2) feature modelling; and 3) classification. We detail 1) and 3) here, and leave 2) for Section 3.

2.1 Pre-Processing

We pre-process text and clean the input as the first step. Text pre-processing is often necessary to standardise token representation. The pre-processing procedures involve both general text and programming language-specific processing.

Parsing and cleaning. PCQA posts contain HTML tags, such as "<font>" and "<a>". We remove these HTML tags and retain only textual contents. We also remove posts with incomplete titles, as they are often not valid posts.

Stop words pruning and tokenisation. We tokenise the sentences using Stanford Parser and prune stop words[5].

Lemmatisation. In natural language text, words exhibit morphological variations (e.g., singular vs. plural, present tense vs. past tense). We normalise the variations by lemmatisation, e.g., converting "inlines" to "inline".

Symbol mapping. For PCQA, posts often have symbols such as "$" (dollar) and "{" (brace) as they contain source code. From preliminary inspection we see questions occasionally use the name of these symbols, e.g., "*What is the purpose of the $ operator in a javascript function declaration?*" vs. "*Can someone explain the dollar sign in Javascript?*". As such, we map all symbols used in programming languages[6] to their symbol names (i.e., "$" to "dollar"). Note that we only map symbols that are tokenised as a single token, so e.g., "div[class=]" will not be converted. Moreover, this conversion is performed only on question titles, as question bodies may contain source code where symbol conversion might not be sensible.

2.2 Binary Classification

We treat the duplication detection task as a binary classification problem. Given a pair of questions, the task is to classify whether they are duplicates or not. The two phases of classification in our work are described as follows.

Classifier training phase. To train a classifier, we first extract a set of features from the question pair. Three types of features are explored: 1) vector similarity; 2) topical similarity; and 3) association score. Feature generation is detailed in Section 3. In total, three features are generated for vector similarity, three feature for topical similarity and one feature for association score. In terms of classifiers, we experiment with the following models: decision tree [5], K-nearest neighbours (K-NN) [2], support vector machines (SVM) [14], logistic regression [26], random forest [15] and naive Bayes [9]. In terms of training data, the ratio of non-

[5]Stanford Parser: `http://nlp.stanford.edu/software/lex-parser.shtml`; stop word list: `http://snowball.tartarus.org/algorithms/english/stop.txt`.

[6]Symbols are collected from: `https://www.tutorialspoint.com/computer_programming/computer_programming_characters.htm`; the set of symbols used: {!, @, #, $, %, ^, &, *, (,), (),+, {, }, {}, >>, .*, _}.

Table 1: Hyper-parameter setting of doc2vec used.

Parameters	Values	Description
size	100	Dimension of word vectors
window	15	Left/right context window size
min_count	1	Minimum frequency threshold for word types
sample	1e-5	Threshold to down-sample high frequency words
negative	5	No. of negative word samples
iter	100	Number of training iterations

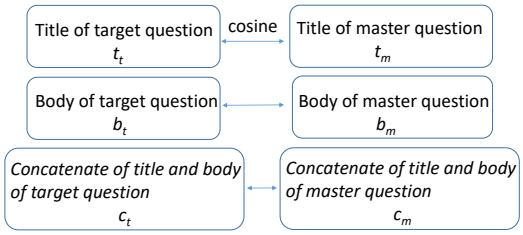

Figure 1: Vector Similarity Features

duplicate pairs (negative examples) to duplicate pairs (positive examples) is very skewed. For example, in the Stack Overflow dump we processed (Section 4.1.1), 716,819 valid posts are tagged with "java" (case-insensitive), among which only 28,656 posts are marked as being duplicates (4.0%). To create a more balanced dataset, we bias the distribution by under-sampling non-duplicate pairs [1, 28]

Duplicates detection phase. Given a question pair (m, t), where m is an existing question and t is a newly posted question, the trained classifier predicts whether they are duplicate. To construct these question pairs, we conduct a naive filtering approach to filter out questions that belong to a different programming language or technique using tags. Tags are mandatory inputs when posting a new question on Stack Overflow, and as such are reliable indicators of the topic the question belongs to. Specifically, we prune existing questions that have no common tags with t, thus narrowing the search space for candidate duplicate questions considerably. We additionally filter out questions that have no answers. We then generate question pairs for t with all remainder questions, and compute features for the classifier to predict the labels.

3. FEATURE MODELLING

We develop three types of features to detect duplicate questions. These features use both surface textual features (Section 3.3) and latent features (Section 3.1 and 3.2) from the questions.

3.1 Vector Similarity

A conventional approach to vectorise text is to compute tf-idf scores for documents. We use tf-idf vectors to compute cosine similarity for a question pair, and the similarity serves as a baseline feature.

In the deep learning community, neural methods for learning word embeddings/vectors have seen plenty of successes for a range of NLP tasks. word2vec, a seminal work proposed by [20], is an efficient neural architecture to learn word embeddings via negative sampling using large corpora of text. Paragraph vectors, or doc2vec, was then introduced to extend word2vec to learn embeddings for word sequences [18]. 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 [17] 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 the paper. doc2vec hyper-parameter settings used in our experiments are detailed in Table 1.

Given target question t (the new question being asked) and master question m (a previously answered question), the task is to classify whether t is a duplicate of m. We generate vectors for the pair's titles (t_m, t_t), body contents (b_m, b_t) and concatenation of titles and body contents (c_m, c_t). We train three doc2vec models, one for each type of vector.

For cosine similarity computation, we compute it between (t_m, t_t), (b_m, b_t), and (c_t, c_m), as illustrated in Figure 1. In total there are three cosine similarity features for each question pair. Despite its simplicity, these features are very effective for duplicate classification.

3.2 Topical Similarity

Latent Dirichlet Allocation (lda) is a well-known implementation of 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. However, 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 [19], which is designed for extracting topics from short texts.

The output of the algorithm is two distributions: the document-topic multinomials and topic-word multinomials. The former describes the distribution of topics in a document; the latter the distribution of words in a topic.

To apply lda, we consider each question as a document. We learn topic models separately for question titles, body contents and the concatenation of both.[7] The topic distribution for a document learnt by lda constitutes its topical vector, and to generate a question pair's topical similarity feature we compute cosine similarity between their topical vectors. In total we have three cosine similarity feature values, one for each topic model.

3.3 Association Score

Manual inspection on duplicate classification using vector and topical similarity features reveals that these features fail to capture certain duplicates. For example, the pair "*Managing several EditTexts in Android*" and "*android: how to elegantly set many button IDs*" is not identified as duplicate. One possible reason could be that these features fail to identify that the phrases "*EditTexts*" and "*button IDs*" are related. This motivates us to look into mining association pairs — pair of phrases that co-occurs frequently in duplicate pairs — to further improve the performance of PCQADup.

[7]We use 30 topics for all topic models.

Table 2: Examples of association pairs mined from Stack Overflow ($\mathcal{A}$) and WikiAnswers.com ($\mathcal{B}$).

Source	Phrase	Associated Phrase
$\mathcal{A}$	append	concatenate
	command prompt	console
	compiled executable	exe
	java notify	java use notifyall
$\mathcal{B}$	the primary language	official language
	a triangular base	a triangle base

We adopt the method developed by [4, 29] which use association pairs to generate features for knowledge base question answering to improve the recall rate. The intuition behind this method is that strong association between elements in target and master questions suggests a high similarity between these two questions. A key of strength of the method is its efficiency: it scales very well with a large amount of data (in our case we consider tens of thousands of questions as potential duplicates for each target question). Note that we use only titles in a question pair to generate association pairs.

After these mining association pairs, we generate association features and lexical features and train a perceptron for duplicate classification. The aim of the exercise is to learn weights for these features. Given a new question pair, we use the trained perceptron to compute the association score for `PCQADup`.

3.3.1 Association Pair Mining

We first extract phrases that co-occur frequently in a duplicate pair in a PCQA corpus. To this end, we process over 25K duplicate questions on Stack Overflow. These questions are manually marked as duplicates by reputable users on Stack Overflow. Note that dataset used for mining association pairs is different to the dataset used for training the association score perceptron and `PCQADup` (dataset details in Section 4.1.1).

For a pair of target question t and master question m, we learn alignment of words between t and m via machine translation [21]. The word alignment is performed in each direction of (m, t) and (t, m) and then combined [16]. Given the word alignments, we then extract associated phrase pairs based on 3-gram heuristics developed in [22]. We prune pairs that occur less than 10 times in the data, and this produces over 130K association pairs. We henceforth refer to this set of association pairs as $\mathcal{A}$.

Additionally, we include another complementary set of 1.3 million association pairs [4], denoted as $\mathcal{B}$, mined from 18 million pairs of question paraphrases from WikiAnswers.com. The rationale for including $\mathcal{B}$ is that it covers a wide range of general phrase pairs that could be relevant to our task. Table 2 presents examples of some association pairs.

3.3.2 Association Score Computation

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 $\mathcal{A} \cup \mathcal{B}$ as the feature value, otherwise it is set to zero. We also generate lexical features for word pairs in t and m, checking for example if they share the same lemma, POS tag or are linked through a *derivation* link on WordNet [13, 4]. The full list of lexical and association features is presented in Table 3.

Take Java related duplicate pairs for example, we generate over 80K association features in total using 5K randomly selected duplicate pairs based on Table 3. We then train a multilayer perceptron with one hidden layer [10] using 5K duplicates and 5K non-duplicates for duplicate classification. As mentioned before the dataset used for generating association features is different to the dataset used for training the perceptron. The aim of the exercise is to learn weights for the 80K features. The weights learnt by the perceptron indicate the predictive power of the features. Features with zero weight are pruned from the feature space. This reduces the number of features to 16K.

After obtaining weights for the features, we compute a weighted combination of the features for a given question pair (the pair is used to evaluate `PCQADup`) to generate a score:

$$score(m, t) = \sum_{i}^{N} v_{f_i} * \theta_{f_i}, \quad (1)$$

where N is the number of features with non-zero weights , v_{f_i} and θ_{f_i} are the value and weight of feature f_i respectively. This score constitutes the association score feature that feeds into `PCQADup`.

4. EXPERIMENT

In this section, we provide analyses to answer several questions: 1) what are the most impactful association features? 2) what is the optimal combination of features for `PCQADup`? 3) how does `PCQADup` perform comparing to the state-of-the-art benchmarks? and 4) how robust is `PCQADup` — does it work for other PCQA domains?

4.1 Experimental Setup

4.1.1 Datasets

Our primary evaluation dataset is Stack Overflow. To test the robustness of `PCQADup`, we additionally evaluate `PCQADup` on other sub-domains of Stack Exchange.

Stack Overflow. The Stack Overflow dataset consists of 28,793,722 questions posted from April 2010 to June 2016[8]. We prune questions without answers, producing 11,846,518 valid questions, among of which 250,710 pairs of questions are marked as duplicates. We use all these duplicate pairs to mine association pairs (described in Section 3.3.1). These association pairs are used in computing association score feature for valid posts of all programming languages (described in Section 3.3.2).

We first focus our evaluation on Java related posts as it has the highest duplicate ratio in the dataset (Section 4.2.4) and obtain 28.6K duplicate question pairs with "Java" tag. As the ratio of non-duplicates to duplicates is very skewed, we keep all duplicates and randomly sample equal number of non-duplicates, thereby obtaining 28.6K non-duplicate question pairs. Our sample thus has 57.2K question pairs or 114.4K questions.

To train the `doc2vec` and `lda` models, we use all Java questions in our sample (114.4K). We could in theory train these

[8] `https://archive.org/details/stackexchange`

Table 3: Lexical and association features used in computing association score. (m, t) is a pair of questions. s_m, s_t are spans in m and t respectively.

Lexical features [4, 29].	
▷ $lemma(s_m) \wedge lemma(s_t)$. $lemma(w)$ is the lemmatised 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?
Association pair counts as feature. $\mathcal{A} \cup \mathcal{B}$ is the set of mined association pairs.	
▷ count of $\langle lemma(s_m), lemma(s_t) \rangle$ if $\langle lemma(s_m), lemma(s_t) \rangle \in (\mathcal{A} \cup \mathcal{B})$, 0 otherwise.	

models using the full data (over 11M questions), but did not do so due to high computation cost.

To create a dataset for the association score experiments, we randomly sample 5K duplicate pairs to generate association pairs, and another 5K duplicate pairs (and 5K non-duplicate pairs) to train the multilayer perceptron.[9]

For `PCQADup` dataset, we use the remainder of 18.6K duplicate pairs (and the same number for non-duplicates) that have not been used. To generate training and test partitions, we split them in the ratio of 4:1.

In addition to Java, we also run experiments for other programming languages. We follow the same procedures as described for dataset generation, except for those with less than 10K duplicate pairs. For these languages, we split them in the ratio of 1:1:3 for association pair generation, multilayer perceptron training and `PCQADup` evaluation respectively. Training and test partitioning remains the same (in the ratio of 4:1).

CQADupStack. CQADupStack provides datasets for 12 sub-forums in Stack Exchange[10]. We select nine sub-forums that have texts of syntactic languages. Table 4 gives the total number of questions and duplicate questions for the selected sub-forums. As the number of duplicates in these datasets is small, we generate only vector similarity and topical similarity features for the question pairs. We train `doc2vec` and `lda` models using all questions for this series of experiments. Training and test split follows the same ratio of 4:1.

Table 4: CQADupStack sub-forum statistics.

Sub-forum	# of duplicates	# of all posts
android	772	23,697
gis	891	38,522
mathematica	865	17,509
programmers	1,020	33,052
stats	670	42,921
tex	3,060	71,090
unix	1,113	48,454
webmasters	529	17,911
wordpress	549	49,146

4.1.2 Evaluation Metrics

We evaluate `PCQADup` using the following metrics:

- Recall and F_1 score. Recall reflects the ability to identify duplicate pairs among the true duplicate pairs. F_1 is the harmonic mean of precision and recall, and is a measure of accuracy.

[9]There is no overlap between the two sets of duplicate pairs.

[10]`http://nlp.cis.unimelb.edu.au/resources/cqadupstack/`

Table 5: F_1 Score of different learning algorithms.

Learning algorithm	F_1 Score
Linear SVM with SGD	0.7825
On-line passive aggressive	0.7330
Perceptron with hidden variables	**0.8475**

- Area under the Receiver Operating Characteristic (ROC) curve (AUC). ROC curve shows how true positive rate changes with respect to false positive rate. AUC computes the probability that our model will rank a duplicate higher than a non-duplicate. Higher AUC score indicates better classification performance.

4.2 Results

We first report the results on association pair mining. Next we analyse classification performance using three categories of features independently and various combinations of them. We experiment `PCQADup` with several classifiers and compare it to state-of-the-art systems on a range of datasets.

4.2.1 Association Feature Performance

To generate the association score for question pairs, we explore three learning algorithms that can asynchronously learn weights for a large number of features: linear SVM with stochastic gradient descent learning [30]; online passive aggressive [12]; and percepton with one hidden layer [10][11].

Table 5 presents the performance of these algorithms in terms of classification accuracy. Perceptron achieves the best accuracy (boldface); we thus use the weights learned by perceptron to generate the association score feature.

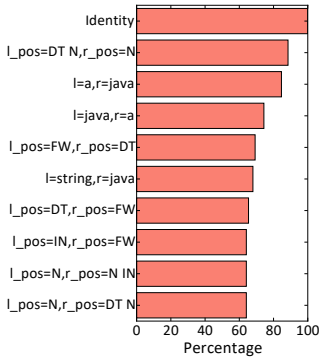

Figure 2: Relative importance of top 10 features.

[11]We use scikit-learn's implementations for these models: `http://scikit-learn.org/stable/index.html`.

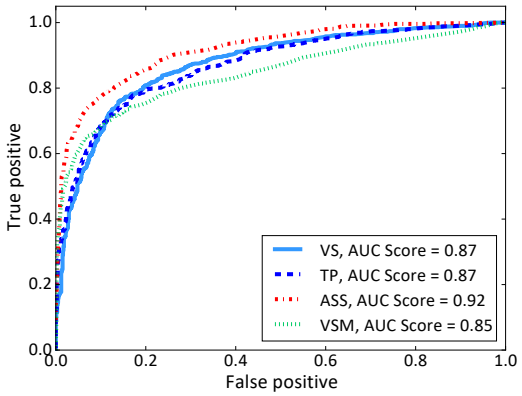

Figure 3: Feature analysis independently.

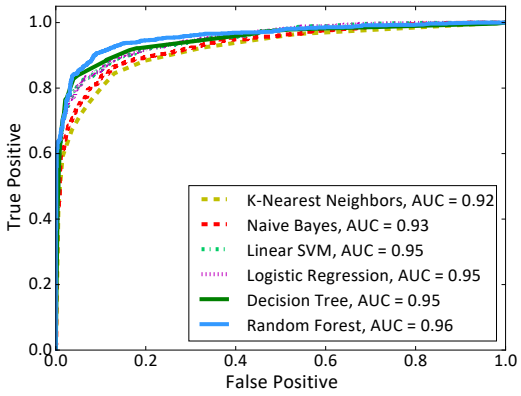

Figure 4: ROC curve on different classifiers on PCQADup (ASS+VS+TP)

To give a sense of the learnt features, we list the top-10 features in Figure 2. In terms of notations, l (left) and r (right) denote the two entities of a pair. For example, *l_pos=DT N, r_pos=N* is a lexical feature, in which the left entity has a POS tag of *DT N*; and the right entity *N*. *Identity* is another lexical feature, and indicates whether the entities have the same lemmatised form. *l=a, r=java* is an association pair feature, for the phrase pair *a* and *java*.

4.2.2 Feature Analyses

We analyse classification performance of PCQADup using following features: doc2vec vector similarity (VS), topical similarity (TP) and association score (ASS). We compare the performances of these features and their combinations with baseline tf-idf document vectors (VSM). We use random forest as the binary classifier in this experiment. Figure 3 presents the ROC curve and AUC score when these features are used independently. We see that ASS performs the best, while VS and TP are slightly worse and VSM has the lowest performance. The strong performance of ASS indicates that co-occurring phrase pairs are an important signal to detect duplicates. The low performance of VSM shows that the pure frequency-based method is not effective in short PCQA posts.

Classification recall and F_1 score using different combinations of features are summarised in Table 6. As with the AUC scores, ASS is the most important feature when used independently. We see that VSM performs substantially worse compared to VS, suggesting that neural embeddings are better document representations. The combination of VS and TP does not outperform ASS by itself, indicating the strength of ASS. Combining association score with either vector similarity (ASS+VS) or topical similarity (ASS+TP) produces similar performance. Encouragingly, when we combine all features (ASS+VS+TP) it yields the best performance, achieving over 0.90 in terms of F_1.

Table 6: Comparison among different feature combinations in PCQADup and VSM

Features	Recall	F_1 Score
VSM	0.6570	0.7599
VS	0.7390	0.7845
TP	0.7430	0.7892
ASS	0.7760	0.8251
VS+TP	0.7450	0.8187
ASS+VS	0.8330	0.8695
ASS+TP	0.8490	0.8735
ASS+VS+TP	**0.8700**	**0.9067**

4.2.3 Classification Algorithms

We measure the classification performance using six classifiers: decision tree, K-nearest neighbours (K-NN), linear SVM, logistic regression, random forest and naive Bayes. The basic parameters used in these classifiers are as follows: We set the maximum depth of the tree to five and use Gini impurity to measure the tree split in decision tree. We use $K = 5$ for K-nearest neighbours and weight each neighbourhood equally. In SVM, we use linear kernel. For logistic regression, we use l_2 penalty. In random forest, the number of trees is set to 10 and the maximum depth of the tree is five. For naive Bayes we use a Gaussian naive Bayes classifier.

Figure 4 presents the ROC curves and AUC scores. From the figure, we can see that our features perform very well with all six classifiers with minor differences. K-NN and naive Bayes are only marginally worse than decision tree, SVM and logistic regression in terms of AUC scores, and random forest has the best performance.

Table 7 presents recall and F_1 score performance for the classifiers. Here we see that random forest has the best performance in both recall and F_1 score. Logistic regression gives the second best recall, followed by decision tree, linear SVM and naive Bayes. K-nearest neighbours performs worst over the two metrics.

Table 7: Comparison among different classifiers on PCQADup (ASS+VS+TP)

Features	Recall	F_1 score
K-Nearest Neighbors	0.7590	0.8272
Naive Bayes	0.7700	0.8397
Linear SVM	0.8210	0.8725
Logistic Regression	0.8340	0.8747
Decision Tree	0.8280	0.8870
Random Forest	**0.8720**	**0.9013**

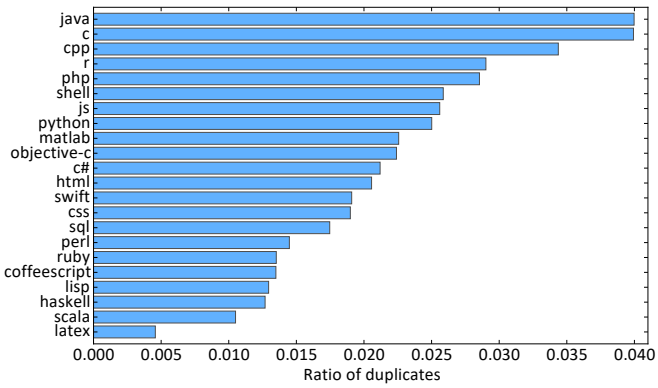

Figure 5: Duplicate distribution of different programming languages (top-22 ranked by ratio.)

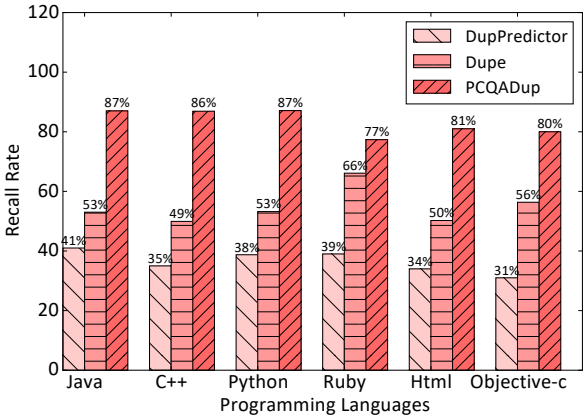

Figure 6: Comparison to the state-of-the-art systems.

The strong performance of random forest is consistent with other findings in the literature [8]. As random forest is an ensemble learning method that constructs a number of decision trees, it is not surprise that it outperforms single-method classifiers.

For K-NN, it assumes each feature has equal importance and its poor performance could be due to this (likely incorrect) assumption for our task. In light of these results, we use random forest as the classifier for all remaining experiments.

4.2.4 Other Programming Languages

We collected some duplicate statistics for 22 most popular programming languages on Stack Overflow. To aggregate different versions of a particular language, we collapse tags of different versions into one tag, e.g., "html" and "html5" tags are collapsed into "html". Figure 5 illustrates the ratio of duplicates to non-duplicates for these programming languages, and Java has the highest ratio of duplicates.

We compare `PCQADup` to state-of-the-art `DupPredictor` [31] and `Dupe` [1] on PCQA duplicate detection over seven programming languages. Performance values of `DupPredictor` and `Dupe` are taken from [1].[12] Figure 6 presents results over duplicate classification recall rate. For all seven languages, `PCQADup` outperforms `DupPredictor` and `Dupe` by at least 11% (Ruby). The biggest improvement is on C++, where the gain is over 35%. The significant gap in terms of performance highlight the strength of `PCQADup`.

4.2.5 Performance on CQADupStack

As the dataset is much smaller compared to Stack Overflow, we use only the VS and TP features and do not generate the association pair features. For comparison, we use the unsupervised baseline proposed in [17], which uses one feature: cosine similarity between c_t and c_m vectors.

Figure 7 shows the performance of `PCQADup` on nine selected sub-forums in CQADupStack over AUC score. We compare two settings of `PCQADup`: using only VS feature, and using both VS and TP features. Note that VS is very similar to the baseline, except that it has two more features: (t_t, t_m) and (b_t, b_m) cosine similarities.

From the figure, we can see that the performance of `PCQADup` is close to the baseline feature. The close gap between the baseline and `PCQADup` (VS) implies the addition of (t_t, t_m) and (b_t, b_m) similarity feature has little impact to the classifier. Adding topical feature (`PCQADup` (VS+TP)) also does not add much to the model in most cases except for the android and mathematica sub-forums. One possible reason for the ineffectiveness of TP and the addition (t_t, t_m) and (b_t, b_m) similarity features in VS is that there are very limited number of duplicate posts in CQADupStack datasets, hence it is hard to learn topics and learn document embeddings.

[12]Technically, the datasets used by `PCQADup` and `DupPredictor` and `Dupe` are not exactly the same. To maximise comparability, we use the duplicate pairs of [1] and sample non-duplicate pairs randomly (these were not specified in the paper).

5. RELATED WORK

Our work is related to previous studies in two fields: 1) question retrieval from QA communities; and 2) Mining PCQA websites.

Question retrieval from QA communities. Large community question-answering data enabled studies to automatically retrieve response of existing questions to answer newly issued questions [6, 7, 23, 27, 28, 32]. Cao et al. [6, 7] 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. Shtok et al. [23] treated the question-answering task as a similarity identification problem, and retrieved questions that have similar titles as candidates. Wang et al. [27] identified similar questions by assessing the similarity of their syntactic parse trees. Yang et al. [28] classified questions based on heuristics and topical information to predict whether they will be answered. Zhou et al. [32] studied synonymy and polysemy to measure question similarity and built a concept thesaurus from Wikipedia for the task. A key difference of our work from these is that we are interested in identifying duplicates specifically for the PCQA domain.

Mining PCQA websites. There has been plenty of research dedicated to mining information from PCQA to assist software development [1, 3, 11, 25, 31]. Ahasanuzza-

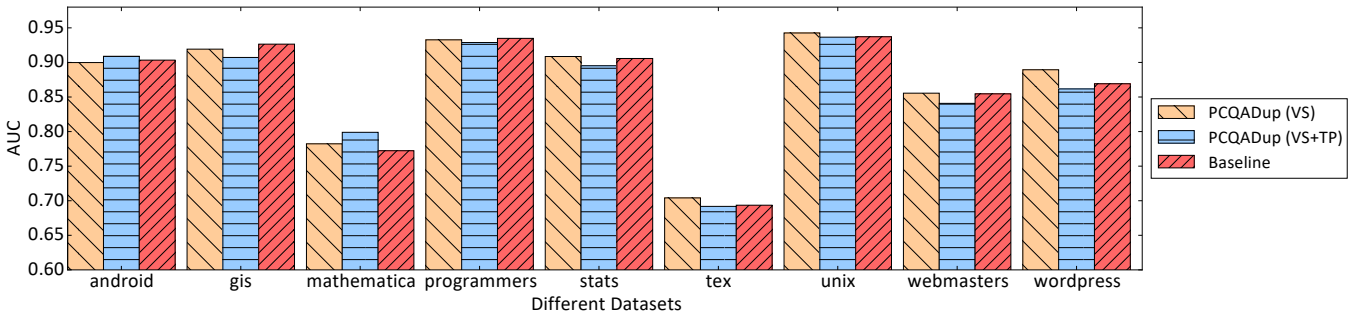

Figure 7: Performance of PCQADup on CQADupStack datasets

man et al. [1] conducted an extensive analysis to understand why duplicate questions are created on Stack Overflow, and trained a binary classification model on question pairs to identify duplicate questions. Their work adopted features from [24] that mine duplicates from Twitter posts using textual and semantic features. Prior to [1], Zhang et al. [31] proposed to classify duplicate questions using title, description, topic and tag similarity as features. It was one of the first studies on duplicate detection in the PCQA domain. To better understand traits and behaviours of Stack Overflow users, Bazelli et al. [3] experimented with a linguistic inquiry and word count tool to analyse text written by users to predict their personalities and categorise them into several groups. Treude et al. [25] analysed the types of questions posted on Stack Overflow and examined which kind of questions are well answered and which ones are unanswered. Correa et al. [11] studied the characteristics of deleted questions on Stack Overflow to predict the deletion of a newly issued question.

6. DISCUSSION AND CONCLUSION

We introduce PCQADup, a new state-of-the-art duplicate detection system for the PCQA domain. PCQADup is driven by a few features, derived using methods from the deep learning and machine learning literature. Combining all features in a classification model, PCQADup outperforms state-of-the-art duplicate detection systems by over 30% in multiple programming languages.

As a product of the association score 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 PCQADup, there is room for improvement. We present some cases (and their reasons) where PCQADup did not detect them as duplicates in Table 8. For example, Q_1 asks for solution for a specific problem "causes of java.lang.NoSuchMethodError". However, question issuer for Q_2 has problem related to "java.lang.NoSuchMethodError" but does not phrase explicitly in the question. Q_3 is phrased in a very specific manner, and has little word overlap with Q_4. Q_6 asks for functions in JQuery that is similar to "isset" in PHP which crosses the programming language boundary. We leave the development of techniques that tackle cross-language duplicate detection for future work.

Besides, the performance of association score feature and topical similarity feature can be further improved if we mine association pairs for different programming languages separately and we learn topics from all the posts instead of only duplicate and equal number of non-duplicate questions. We leave these improvements as future work.

Table 8: Examples of duplicate detection exceptions

Q_1	Causes of 'java.lang.NoSuchMethodError'
Q_2	How should a minimal Java program look like?
Reason:	**Implicit expression**
Q_3	What does (function (x,y)...)(a,b); mean in JavaScript?
Q_4	javascript syntax explanation
Reason:	**Difficult paraphrases**
Q_5	Finding whether the element exists in whole html page
Q_6	isset in Jquery?
Reason:	**Cross-language duplicates**

7. ACKNOWLEDGMENTS

We would like to thank Pengcheng Yin (CMU) and Wenjie Ruan (University of Adelaide) for their feedback on the paper.

8. REFERENCES

[1] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider. Mining Duplicate Questions in Stack Overflow. In *Proc. of the MSR 2016*, pages 402–412.

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

[3] B. Bazelli, A. Hindle, and E. Stroulia. On the Personality Traits of StackOverflow Users. In *Proc. of the ICSM 2013*, pages 460–463.

[4] J. Berant and P. Liang. Semantic Parsing via Paraphrasing. In *Proc. of the ACL 2014*, pages 1415–1425.

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

[6] X. Cao, G. Cong, B. Cui, and C. S. Jensen. A Generalized Framework of Exploring Category Information for Question Retrieval in Community Question Answer Archives. In *Proc. of the WWW 2010*, pages 201–210.

[7] X. Cao, G. Cong, B. Cui, C. S. Jensen, and Q. Yuan. Approaches to Exploring Category Information for Question Retrieval in Community Question-Answer Archives. *ACM Transactions on Information Systems*, 30(2):7, 2012.

[8] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proc. of the ICML 2006*, pages 161–168.

[9] T. F. Chan, G. H. Golub, and R. J. LeVeque. Updating Formulae and A Pairwise Algorithm for Computing Sample Variances. In *Proc. of the 5th Symposium in Computational Statistics (COMPSTAT 1982)*, pages 30–41, 1982.

[10] M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proc. of the EMNLP 2002*, pages 1–8.

[11] D. Correa and A. Sureka. Chaff from the Wheat: Characterization and Modeling of Deleted Questions on Stack Overflow. In *Proc. of the WWW 2014*, pages 631–642.

[12] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[13] C. Fellbaum. WordNet: An Electronic Lexical Database. *MIT Press*, 1998.

[14] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support Vector Machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[15] T. K. Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[16] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proc. of the NAACL 2003*, pages 48–54.

[17] J. H. Lau and T. Baldwin. An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. In *Proc. of the RepL4NLP 2016*, pages 78–86.

[18] Q. V. Le and T. Mikolov. Distributed Representations of Sentences and Documents. In *Proc. of the ICML 2014*, pages 1188–1196.

[19] C. Li, H. Wang, Z. Zhang, A. Sun, and Z. Ma. Topic Modeling for Short Texts with Auxiliary Word Embeddings. In *Proc. of the SIGIR 2016*, pages 165–174.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Proc. of the NIPS 2013*, pages 3111–3119.

[21] F. J. Och and H. Ney. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51, 2003.

[22] F. J. Och and H. Ney. The Alignment Template Approach to Statistical Machine Translation. *Computational Linguistics*, 30(4):417–449, 2004.

[23] A. Shtok, G. Dror, Y. Maarek, and I. Szpektor. Learning from the Past: Answering New Questions with Past Answers. In *Proc. of the WWW 2012*, pages 759–768.

[24] K. Tao, F. Abel, C. Hauff, G. Houben, and U. Gadiraju. Groundhog Day: Near-Duplicate Detection on Twitter. In *Proc. of the WWW 2013*, pages 1273–1284.

[25] C. Treude, O. Barzilay, and M. D. Storey. How Do Programmers Ask and Answer Questions on the Web? In *Proc. of the ICSE 2011*, pages 804–807.

[26] S. H. Walker and D. B. Duncan. Estimation of the Probability of an Event as a Function of Several Independent Variables. *Biometrika*, 54(1-2):167–179, 1967.

[27] K. Wang, Z. Ming, and T. Chua. A Syntactic Tree Matching Approach to Finding Similar Questions in Community-based QA Services. In *Proc. of the SIGIR 2009*, pages 187–194.

[28] L. Yang, S. Bao, Q. Lin, X. Wu, D. Han, Z. Su, and Y. Yu. Analyzing and Predicting Not-Answered Questions in Community-based Question Answering Services. In *Proc. of the AAAI 2011*, pages 1273–1278.

[29] P. Yin, N. Duan, B. Kao, J. Bao, and M. Zhou. Answering Questions with Complex Semantic Constraints on Open Knowledge Bases. In *Proc. of the 24th ACM International on Conference on Information and Knowledge Management, (CIKM 2015)*, pages 1301–1310, October 2015.

[30] T. Zhang. Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms. In *Proc. of the ICML 2004*, pages 919–926, 2004.

[31] Y. Zhang, D. Lo, X. Xia, and J. Sun. Multi-Factor Duplicate Question Detection in Stack Overflow. *Journal of Computer Science and Technology*, 30(5):981–997, 2015.

[32] G. Zhou, Y. Liu, F. Liu, D. Zeng, and J. Zhao. Improving Question Retrieval in Community Question Answering Using World Knowledge. In *Proc. of the IJCAI 2013*, pages 2239–2245.