2018 25th Asia-Pacific Software Engineering Conference (APSEC)

Detecting Duplicate Bug Reports with Convolutional Neural Networks

Qi Xie*, Zhiyuan Wen§, Jieming Zhu‡, Cuiyun Gao¶, Zibin Zheng§

*School of Computer Science and Technology
Southwest Minzu University
Chengdu, China
qi.xie.swun@gmail.com

§School of Data and Computer Science
Sun Yat-sen University
Guangzhou, China
wenzhy7@mail2.sysu.edu.cn
zhzibin@mail.sysu.edu.cn

‡Huawei 2012 Labs
Shenzhen, China
jmzhu@ieee.org

¶Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong, China
cygao@cse.cuhk.edu.hk

***Abstract*—Bug tracking systems are widely used to track bugs from users during the lifecycle of software systems for reliability maintainence. When software systems have a large user base, which is common in practice, different users may encounter a same bug and then generate many duplicate bug reports. In a large project, each bug report is usually assigned to a different developer or team to parallelize the bug debugging and fixing activities. The presence of duplicate bug reports thus leads to many unnecessary efforts of developers spending on debugging a same issue. To speed up the bug fixing process and save the cost of developers, there is a high demand for automated detection of duplicate bug reports. In this paper, we explore the use of powerful deep learning techniques, including word embedding and Convolution Neural Networks, to calculate the similarity between a pair of bug reports and thus identify possible duplicates. In contrast to previous work that consider only common words between bug descriptions for lexical similarity computation, our approach is able to better capture semantic similarity between words. We further improve traditional CNN models by combining some domain-specific features extracted from bug reports. Evaluation results on the bug reports from four popular open-source projects show that DBR-CNN has made a significant improvement on duplicate detection accuracy over traditional approaches.**

***Keywords*-Software reliability; bug reports; duplicate detection; deep learning; CNN**

I. Introduction

Modern software systems are not only growing fast in complexity and size, but also keep updating frequently under the agile development mode. It is a common case that a software system has tens of components, built by hundreds of developers. It is thus impossible to guarantee every release of a complex software system is bug-free. Developers usually employ a bug tracking system (e.g., Jira, Bugzilla) to file bugs found by testers and issues encountered by users. Fixing open bugs has become a daily job for developers to aid in improving reliability of software systems.

However, it is a pain for developers when too many duplicate issues are reported [1], [2]. For example, the average ratio of duplicate issues reaches about 20% in our studied systems as shown in Table. I[1]. There are two main reasons for such duplication of bug reports: 1) Popular software systems usually have up to millions of users. Different users or testers may encounter the same issue in totally different ways. The large use base can lead to repeated triggers of a same bug. 2) As software systems update new versions frequently, multiple versions of software systems for the same application may co-exist. Then, users may have different visions of software systems installed. A resolved issue in higher version may be encountered again in other lower version. All such duplicated issues, if not detected, it will increase the debugging cost of developers. For a large software system, it is not uncommon that duplicate issues are assigned to different developers or teams to resolve. Due to its great importance, detection of duplicate bug reports has attracted much attention from both researchers and practitioners [2]–[6].

Although the use of bug tracking systems helps a lot in the whole process from bug reporting to bug fixing, there are still some challenges to be addressed in terms of duplicate bug report detection. Bug tracking systems assist with bug report collection, but it by now provides no support for automated duplicate detection. Hence, the *triager* needs to manually inspect whether a bug report is a duplicate or not [6]. This process is not only time-consuming and error-prone, but also enhances the cost of software maintenance [5]. Moreover, the large number of bug reports makes it impractical for a triager to manually handle so many duplicate bug reports. As a result, automated detection of duplicate bug reports is urgently in need.

In order to achieve this goal, some studies are devoted to automating the detection process, which mainly include two ways. The first one is employing the traditional Natural

[1]https://github.com/LogPAI/bugrepo

TABLE I
DUPLICATE BUG IDENTIFICATION

Project	Timespan	#Components	#Issues	#Issue/day	#Duplicates	%Duplicates	Median Resolving Time
Mozilla Core	1997/03/28 ∼2013/12/31	130	205,069	33.5	44,691	21.8%	102.1 days
Firefox	1999/07/30 ∼2013/12/31	52	115,814	22.0	35,814	30.9%	76.4 days
Thunderbird	2000/04/12 ∼2013/12/31	23	32,551	6.5	12,501	38.4%	83.7 days
Eclipse Platform	2001/10/10 ∼2013/12/30	21	85,156	19.1	14,404	16.9%	29.8 days
JDT	2001/10/10 ∼2013/12/31	6	45,296	10.1	7,688	17.0%	23.0 days
Spark	2010/04/01 ∼2018/01/10	29	22,639	8.0	3,077	13.6%	7.1 days
Hadoop	2005/07/24 ∼2017/11/01	45	12,855	2.9	1,861	14.5%	14.3 days
MapReduce	2006/03/17 ∼2018/01/15	63	7,019	1.6	977	13.9%	28.2 days
Hdfs	2006/04/06 ∼2018/01/12	71	12,779	3.0	1,659	13.0%	9.7 days
HBase	2007/02/27 ∼2018/01/21	95	19,788	5.0	1,340	6.8%	6.8 days
Cassandra	2009/03/07 ∼2018/01/21	24	14,071	4.3	2,083	14.8%	8.6 days
Mesos	2011/02/16 ∼2018/01/26	40	8,454	3.3	800	9.5%	23.5 days

TABLE II
EXAMPLES OF BUG REPORTS FROM JIRA

ID	SPARK-22888	SPARK-22899
Type	Bug	Bug
Priority	Critical	Major
Component	ML	ML; Structured Streaming
Title	OneVsRestModel does not work with Structured Streaming	OneVsRestModel transform on streaming data failed.
Description	OneVsRestModel uses Dataset. persist; which does not work with streaming. This should be avoided when the input is a streaming Dataset.	OneVsRestModel transform on streaming data failed. Because of it persisting the input dataset; which streaming do not support.
Status	Resolved	Resolved
Resolution	Duplicate	Fixed
Duplicated_issue	SPARK-22899	SPARK-22888
Assignee	Unassigned	Weichen Xu
Reporter	Joseph K. Bradley	Weichen Xu
Created_time	2017/12/22,22:38:00	2017/12/25,10:20:00
Affects_version	2.2.1	2.2.1
URL	https://issues.apache.org/jira/browse/SPARK-22888	https://issues.apache.org/jira/browse/SPARK-22899

Language Processing (NLP) and Information Retrieval (IR) techniques to help with duplicate detection [6], [7]. However, in such a way, when a new bug report is received, it needs to choose key words to search in bug tracking systems. The other one is to perform classification by employing a threshold to determine whether the newly-reported bug report is duplicate or not [8]–[10]. Although these approaches can ease the pressure of triager partially, the accuracy of duplicate detection is still far from satisfaction. Both ways of the above approaches, however, do not fully take into account the semantic information in bug report descriptions. Recently, Deep Learning (DL) techniques are employed to enhance the performance of NLP in IR [11], [12], since DL techniques can effectively extract complicated non-linear features.

Motivated by the above intuition, we explore the following problems:

- Can we exploit DL techniques in automated detection of duplicate bug reports to improve the detection performance?
- Can we take advantages of both syntactic and semantic features for automated detection of duplicate bug reports by using DL?

In this paper, we propose a framework named detecting Duplicate Bug Reports with Convolutional Neural Network (DBR-CNN), which extracts both syntactic and semantic features from bug reports. In summary, the contributions of this paper are three-fold:

- We construct a CNN-based framework to automate the detection process of duplicate bug reports, which obtains both syntactic and semantic information of bug reports.
- We combine the CNN-trained features and some domain-specific features (e.g., component, bug serverity, issue time), which benefits the automated detecting of duplicate bug reports.
- We evaluate our approach on four bug report datasets collected from popular open-source projects, and the results show that DBR-CNN is effective in enhancing the detection performance and the proposed DBR-CNN is also extensible and flexible.

The rest of this paper is organized as follows. Section II presents the background of duplicate bug reports. Section III details our approach to detect duplicate bug reports with convolutional neural network. Section IV shows the experimental results. We discussion the limitations of our work in Section V. Section VI describes the related work and Section VII concludes the paper.

II. BACKGROUND

In this section, necessary background information is introduced. We first elaborate styles of bug reports and duplicate bug reports, then present word embedding, finally describe the Convolutional Neural Network (CNN).

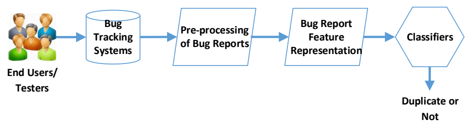


Fig. 1. Detecting duplicate bug reports process

A. Bug Report

A bug report is a structured record consisting of several fields [6], which can be collected by bug tracking systems. As shown in Table. II, there are examples of two bug reports from Jira[2]. It lists the primary fields which denote different types of bug information, such as *ID*, *type*, *priority*, *component*, *title*, *description*, *status*, *resolution*, *duplicated_issue*, *assignee*, *reporter*, *affects_version*, *URL* and so forth. Specifically, field *title* is a summary of the defect problem and the field *description* is the elaborate description of how the defect occurred. Fields *title* and *description* are described using natural language type named *textual features*, while other fields such as *status* and *resolution* are presented with other perspectives named *domain-specific features* [7]. In this paper, we have collected four types of bug reports from Jira to validate our approach, which include hadoop, hdfs, mapreduce and spark.

B. Detecting Duplicate Bug Reports

Table. II shows an example of a pair of duplicate bug reports with *ID SPARK-22888* and *SPARK-22899* respectively. We can see that as to the same defect of OneVsRestModel, two bug reports are described with different words and syntaxes from different reporters Joseph K. Bradley and Weichen Xu. If a new incoming bug report is duplicate such as *SPARK-22888*, then it will be marked as "*duplicate*". To reduce the time consumption and enhance the efficiency of bug handle, it should be detected whether these bug reports have been submitted before or not when same bug reports are incoming.

Fig. 1 illustrates a typical automatical process of detecting duplicate bug reports [2], [6], [7], [13]. Firstly, when end users or testers encounter software defects, the defects information will be reported to bug repository by employing bug tracking systems. Secondly, due to the multiple fields, the bug reports should be preprocessed which mainly includes three steps. In addition, the bug report feature can be extracted by transforming words into representation which would capture the syntactic and semantic information [14]. After that, classifiers can detect whether the new incoming bug report is duplicate or not by using the feature representation.

C. Word Embedding

Word embedding[3] is the collective name for a set of language modeling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers. There exists some work focusing on word embedding which includes Neural Network Language Model (NNLM), Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA), CBOW, Skip-gram [15]–[18]. Specially, Mikolov et al. [18] publish the codes of continuous bag-of-words and skip-gram architectures for computing the word vectors, which not only enhance the word vectors quality but also reduce the computational complexity[4].

Thus, the learning high-quality word vectors can be used to detect the similar words or other fields, such as questions, issue reports, and bug reports, by finding the word vectors similarity. In our work, word embedding is employed after the pre-processing step in which the skip-gram neutral network is adopted.

[2]https://jira.atlassian.com/
[3]https://en.wikipedia.org/wiki/Word_embedding

D. Convolutional Neural Network

CNNs are feed-forward multilayer neural networks, which are constructed by one or more convolutional and subsampling layers with fully connected layers optionally. CNNs have successfully been employed to ID-grid of time-series data and 2D grid of image Data [19]. Furthermore, CNNs are also widely adopted in many practical domains such as speed recognition [20], image classification [21] and NLP [14]. In this paper, we employ a CNN to extract the syntactic and semantic feature vectors as shown in Fig. 3 which will be elaborated in section III.

III. Approach

In this section, we demonstrate our proposed approach DBR-CNN as shown in Fig. 2. First we introduce the data preprocessing and word embeddings. And then, we employ CNN to represent bug reports. Finally, CNN-trained features and domain-specific features are combined to enhance the performance of automated detection, where CNN training procedure is also presented.

A. Data Preprocessing

As shown in Table. II, bug reports include much textual information, then a standard NLP process is employed in our approach as the first step. Generally, there are three steps of data preprocessing: [2], [22]

- Parsing tokens: The words in the textual fields of bug reports will be parsed (i.e., title etc.). And the parsed textual fields will be turned to a stream of characters, in which such as capitals, punctuation, brackets etc will be removed. A word can be regarded as a token.
- Stemming: The target of stemming is to discover the stem of words without multiple grammatical forms. For example, the words *does* and *structured* are converted to *do* and *structure* respectively of *SPARK-22888* in Table II.
- Removing stop words: Same words that do not carry significant information will be removed, i.e., *the*, *a*, and *that* etc.

[4]https://code.google.com/p/word2vec/

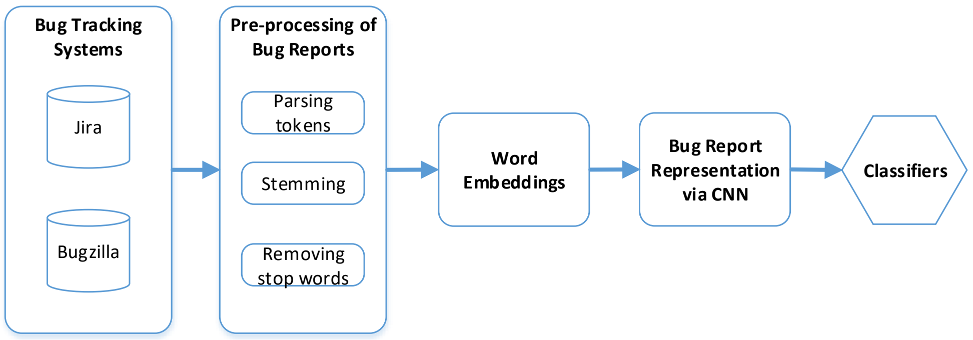


Fig. 2. Overall workflow for our approach

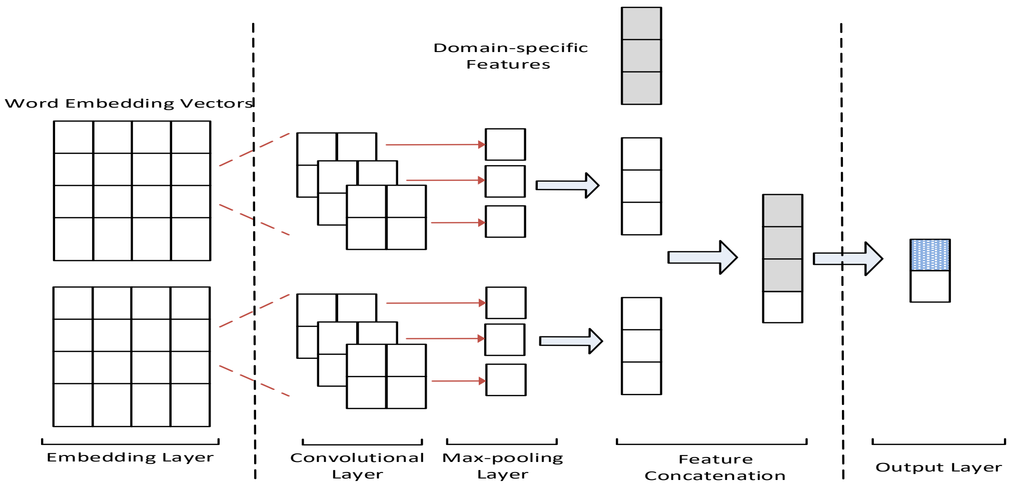


Fig. 3. Bug report representation via CNN

B. Word Embeddings

As shown in Fig. 2, word embeddings are employed to transfer tokenized words into real-valued feature vectors by using unsupervised pre-training [14]. Here, we employ randomly-generated embeddings in comparison with baselines and word2vec in other comparisons. Detailed discussion of the embedding layer can be found in Section IV.

C. Bug Report Representation via CNN

As detailed in Fig. 3, we employ word embeddings as the input layer, which provide real-valued feature vectors to the convolutional layer. Then, the convolutional layer and max-pooling layer are used to build two distributed vector representations of any two input bug reports. After that, we can obtain the similarity score (*i.e.*, textual features) of two input bug reports. In addition, domain-specific features (i.e., Component, Create_time and Priority etc.) and textual features are concatenated. Finally, the concatenated vectors are put into the output layer with logistic regression for classification.

1) CNN Architecture: Owing to the different sizes of bug reports, we use d to represent the dimensionality of the word vectors. Also we employ s to denote the length of an input bug report, so we can obtain an $s \times d$ matrix of a bug report in which rows express distributed tokens [23].

Given a weight vector $w \in \mathbb{R}^{h \times d}$ where h represents the region size, there are $h \times d$ parameters to be trained. we suppose the bug report matrix by $U \in \mathbb{R}^{s \times d}$, and then, employ $U[i : j]$ to denote the sub-matrix of U from row i to row j. The convolutional layer iteratively employs the filter on the sub-matrices of U as follow:

$$v_i = w \cdot U[i : i + h - 1], \qquad (1)$$

where $i = 1...s - h + 1$, and $\cdot$ denotes the dot product. In addition, the length of v is $s - h + 1$. To obtain the feature maps of bug reports with CNN, a bias term and an activation function are employed as follows:

$$g_i = f(v_i + b), \quad (2)$$

where b denotes the bias with $b \in \mathbb{R}$ and f is the activation function with hyperbolic tangent function.

2) Combining Domain-specific Features: From (2), we can obtain the feature maps (g_1, g_2) of the input pair of bug reports (r_{g1}, r_{g2}). And the cosine similarity of (r_{g1}, r_{g2}) is defined as:

$$Sim(r_{g_1}, r_{g_2}) = \frac{g_1 \cdot g_2}{\|g_1\| \, \|g_2\|}. \quad (3)$$

In our model DBR-CNN, we enrich the textual features by concatenating corresponding domain-specific features, which benefits the automated detecting of duplicate bug reports. Thus, the similarity of two CNN-trained feature maps of bug reports pairs can be combined to domain-specific features of bug reports. Then, the concatenated vector is defined as follows:

$$c = [Sim \quad Component \quad Create\_time \quad Priority], \quad (4)$$

where the concatenated vector c is fed into the output layer with logistic regression classifier as shown in Fig. 3.

IV. Experiments

A. Data

In this paper, four types of bug reports from Jira bug tracking system are collected, which include hadoop, hdfs, mapreduce and spark. Table III shows the summary of the four datasets, which are from different applications. As shown in Table III, we employ the bug report datasets reported between July 2005 to January 2018. In addition, the bug report datasets hadoop, hdfs, mapreduce and spark include 1,840, 12,779, 7,019, 22,639 bug reports respectively, in which the numbers of duplicate bug reports are 228, 1,569, 957, 2,963 severally.

What's more, the number of duplicate buy reports will be further reduced owe to some special cases. Different from Sun et al. [7] choose the first *M* reports in the repository of which 200 reports are duplicates as training. In this work, for example, the bug report dataset Hdfs includes 12,779 bug reports and 1,569 duplicate bug reports which are real duplicates. Specially, there may be two pairs where *A* bug report and *B* bug report are duplicate $(A \rightarrow B)$, also *B* bug report and *A* bug report are duplicate $(B \rightarrow A)$. Due to the repeats of duplicate bug reports, we only choose one of them according to the issue ID. In addition, the bug reports in which the issue IDs in URL are not consistent with their real ID will also be removed. Furthermore, the selected bug reports should have the duplicate_issue filed information without empty. Thus, the number of duplicate bug reports decreases from 1,569 to 715 in Hdfs dataset.

Since the number of duplicate bug reports is much less than the number of non-duplicate bug reports, the imbalance of data should be handled. To address this problem, a pair of two bug reports which includes two duplicate bug reports is defined as positive sample, while a pair of two bug reports which includes two non-duplicate bug reports is defined as negative sample. In this paper, we set the number of positive samples as the same as the number of negative samples approximately. We extract positive samples from each dataset deleting overlaps and invalid bug reports. Table IV shows that the total samples are divided into three parts randomly, one part as the training set which covers 70%, the second part as the validation set which is 10%, and the last part as the test set which has 20% for our experiments. For example, we can obtain 715 duplicate bug reports for Hdfs dataset after deleting overlaps and invalid bug reports from 1,569 duplicate bug report as shown in Table III. After that, we construct the number of negative samples as 714 according to the number of positive samples. Then, the number of total instances is 1,429 as shown in Table IV. Thus, we select 1,000 bug reports as the training instances, 143 bug reports as the validation instances, 286 bug reports as the test instances randomly and respectively.

B. Evaluation Measures

Two widely adopted metrics F-measure [19] and Accuracy [11] are employed in this paper to measure the performance of detection.

We follow the notations of *F-measure* definition in work [19], by using the following equations:
Precision (P): The ratio of the number of positive samples correctly classified as duplicate to the number of samples classified as duplicate.
Recall (R): The ratio of the number of positive samples correctly classified as duplicate to the number of positive samples.
F-measure ($F\text{-}measure$): The traditional F-measure is the harmonic mean of precision P and recall R as follows:

$$F\text{-}measure : F = \frac{2 * P * R}{P + R}. \quad (5)$$

And the **Accuracy** is defined as:

$$Accuracy = \frac{Number\ of\ correct\ detections}{Number\ of\ bug\ report\ samples}, \quad (6)$$

the higher the **F-measure** and **Accuracy** are, the better the detection performance presents.

C. Baselines

To study the effectiveness of our DBR-CNN approach, we compare the performance of three typical approaches:

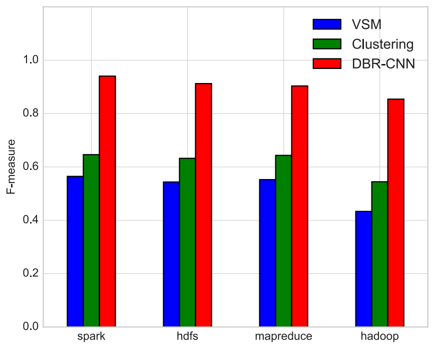
- VSM [2]: The vector space model represents each word in a multi-dimensional space. Each dimension of the space corresponds to a word. The position along each axis in this space depends on the frequency of the word occurring in the text. The similarity between two words is then measured in terms of distances (cosine distance here for its good performance) in this vector space.
- Clustering [5]: Jalbert and Weimer build a linear regression model based on the results of clustering and textual similarity of bug descriptions. The incoming bugs are determined duplicate if they present higher similarities

TABLE III
SUMMARY OF DATASETS

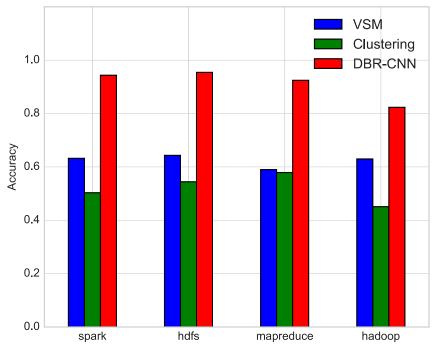
DataSets	All Bug Reports	Duplicate Bug Reports	Time Frame
Hadoop	1840	228	Jul/24/2005-Jan/12/2018
Hdfs	12779	1569	Apr/06/2006-Jan/12/2018
Mapreduce	7019	957	Mar/17/2006-Jan/15/2018
Spark	22639	2963	Apr/01/2010-Jan/10/2018

TABLE IV
DATASETS DIVISION

DataSets	Training Instances	Validation Instances	Test Instances	Total Instances
Hadoop	153	22	44	219
Hdfs	1000	143	286	1429
Mapreduce	526	76	151	753
Spark	2296	328	657	3281



(a) F-measure



(b) Accuracy

Fig. 4. Performance comparison with baselines

to reports in historical data. Specifically, the clustering graph algorithm designed for social networks is applied.

- Traditional CNN: A CNN model is employed directly to classify the subject datasets without considering other features. We use this model as baseline to illustrate the usefulness of domain-specific features in duplicate bug detection.

When comparing with the performance of above approaches with our DBR-CNN approach, we employ the same bug report datasets, preprocessing, tools, and handling data imbalance.

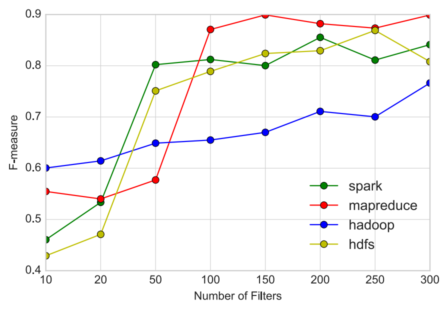
D. Results

We illustrate experimental results of different approaches on the four datasets in this part. And the average values of measures for 10 times are presented.

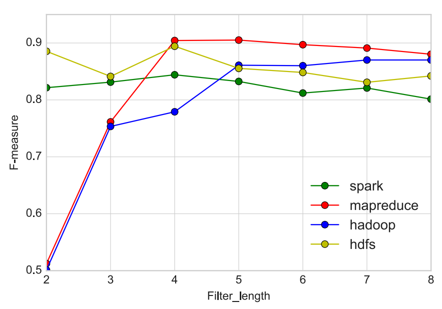
1) Comparison with Baselines: To demonstrate the effectiveness of employing deep models, we compare DBR-CNN with the baseline approaches without deep neural networks involved, *i.e.*, VSM and clustering, shown in Fig. 4. As can be seen, our model can always achieve the best results on the four datasets, with average F-measure and accuracy at 0.903 and 0.919 respectively. Specifically, DBR-CNN model increases VSM by 72% for F-measure and 46% accuracy on average, while improving the clustering approach by 46% and 75% respectively. The remarkable improvement indicates that deep models can better learn the semantic representations of bug reports than non-deep models.

2) Comparison with traditional CNN approach: To verify whether the feature-enhanced model DBR-CNN can improve the performance of original CNN model, we conduct the comparison on the four datasets, with results depicted in Table V.

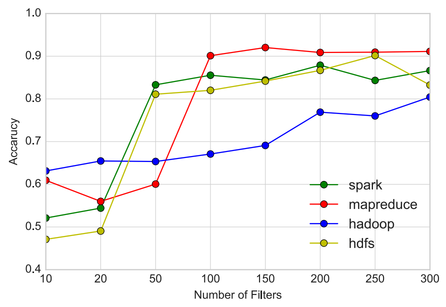
As can be seen, our model consistently outperform the traditional model on all the datasets, increasing by 12.8% for F-measure and 9.9% for accuracy on average. In terms of accuracy, DBR-CNN achieves 0.823, 0.954, 0.925, and 0.944 for hadoop, hdfs, mapreduce, and spark, respectively, enhancing traditional CNN by 3.7%∼18.1%. The obvious improvement demonstrates the effectiveness of enriching textual features with the domain-specific information for detecting duplicate bugs. From further observation, we discover that DBR-CNN greatly improves the performance of traditional CNN on hadoop and mapreduce datasets, with increase rate at 14.9% and 25.4% for F-measure respectively. As Table III and Table IV shown, both datasets have relevantly few available bug reports compared to other two datasets. We therefore



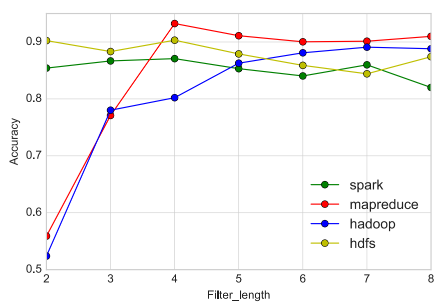
(a) Impact of different number of filters



(b) Impact of different filter length

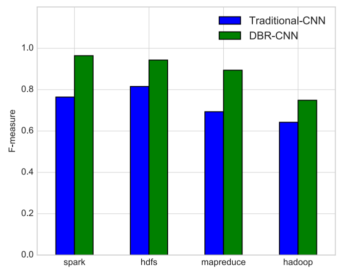


(c) Impact of different number of filters

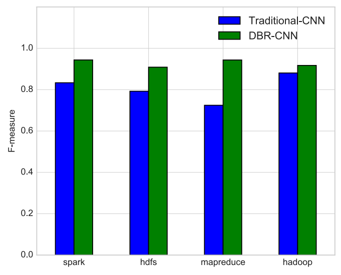


(d) Impact of different filter length

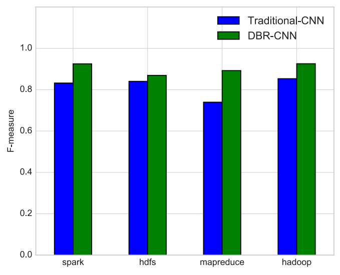
Fig. 5. Performance of DBR-CNN under different parameter settings



(a) Random

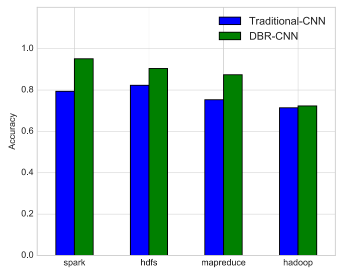


(b) Glove

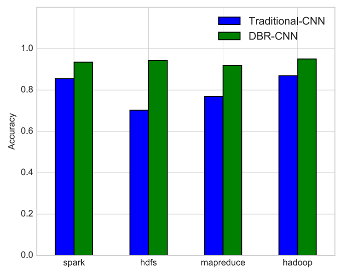


(c) Word2vec

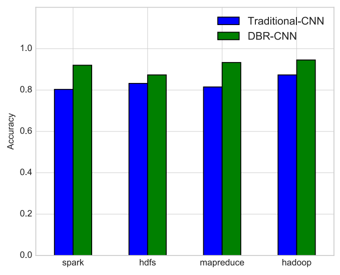
Fig. 6. Performance comparison with F-measure



(a) Random



(b) Glove



(c) Word2vec

Fig. 7. Performance comparison with accuracy

TABLE V
PERFORMANCE COMPARISON WITH TRADITIONAL CNN APPROACH

DataSets	F-measure		Accuracy	
	Traditional	DBR-CNN	Traditional	DBR-CNN
hadoop	0.743	0.854	0.794	0.823
hdfs	0.866	0.912	0.912	0.954
mapreduce	0.720	0.903	0.782	0.925
spark	0.870	0.940	0.857	0.944
Average	0.800	**0.903**	0.836	**0.919**

attribute the low performance of traditional CNN to that the model may fail to capture the textual information with a handful of training data [24]. Overall, DBR-CNN can well modify the traditional CNN with other information, *i.e.*, *component*, *create time*, and *priority*, especially for small datasets.

3) Performance Under Different Parameter Settings: We explore the impact of parameter settings on the performance of DBR-CNN, and the average values of measures for 5 times are given. Fig. 5 illustrates the variations of F-measure (above) and accuracy (below) along with different filter numbers and filter lengths on the four datasets. As Fig. 5 (a) and (c) shown, more filters are beneficial to generate more representative features for bug reports and lead to better performance. Most datasets except hadoop exhibit sheer increase when the filter number range from 20 to 100. When filter number becomes larger than 100, the performance of DBR-CNN tends to stabilize at a high point. For hadoop, its performance increase generally requires much more filter numbers (150 in our case), which may be due to its small training data. Then, the number of filters in our approach is set to 100.

In terms of filter length, depicted in Fig. 5 (b) and (d), longer filter lengths do not signify better performance. For example, both F-measure and accuracy of the datasets hdfs and spark display a steady changes over different filter lengths. However, longer filter lengths can obviously enhance the performance on other two datasets. When setting the filter length larger than five, duplicate bug reports from mapreduce and hadoop can be well identified. Thus, we achieve that longer filter lengths are more helpful for small datasets (*e.g.*, fewer than 500 bug reports). For large-scale datasets, adjustment of filter lengths may not distinctly improve the performance. So, in our experiments, we set the values of filter length as 3, 4, 5 for hadoop dataset and 4, 5, 6 for other three datasets respectively.

4) Performance with Different Word Embeddings: To further validate the extensibility and flexibility of our approach, we continue to compare the performance of our DBR-CNN approach with traditional CNN by using different word embeddings. Three word embeddings are employed in this section which include

- Randomly-generated embeddings: Each word embedding is a random uniform distribution in the range [-0.5, 0.5) and divided by the length of the embedding layer.
- Glove [25]: The word embeddings are trained based on aggregated global word-word co-occurrence statistics from a corpus, provided by Stanford University.
- Word2vec: The word embeddings are fine-tuned based on the subject datasets during the training process, instead of using existing word embeddings. Specifically, we employ the skip-gram neural network architecture to perform pre-training, which is available in the word2vec tool [18]. And four bug report datasets are employed to train word embeddings, including hadoop, hdfs, mapreduce, and spark from Jira.

Fig. 6 and Fig. 7 show the F-measure and accuracy of DBR-CNN approach and traditional CNN approach with three word embeddings respectively. Specially, this two figures demonstrate that our DBR-CNN approach outperforms traditional CNN approach both in F-measure and accuracy with hadoop, hdfs, spark, mapreduce datasets under Random, Glove, Word2vec word embeddings separately. This observation shows that our approach is effective in enhancing the detection performance. What's more, Table VI lists the values of F-measure and accuracy in detail. From the Table VI, we can see that Glove and self-trained word2vec can outperform randomly-generated word embeddings in most cases. This indicates that the performance of our model can be improved by modifying specific components. Furthermore, Table VI also shows that DBR-CNN approach can enhance the detection performance remarkably. Our proposed approach can improve the F-measure by 22%, 15%, and 11% with random, Glove, and word2vec respectively, and also can enhance the accuracy by 12%, 17%, and 10% severally. It demonstrates that our proposed approach not only can improve the detection performance but also has the extensible and flexible characteristics.

V. LIMITATIONS AND DISCUSSION

External Validity: First, our approach is evaluated on four datasets, which may not guarantee the generalization of the proposed approach. We alleviate this threat by using datasets from different platforms. Second, the number of bug reports for training the neural network model may not be sufficient. However, experimental results show that our proposed model can outperform all the baselines, even with few datasets. We will continuously collect more subject bug reports, and release our datasets for publicly available in future.

Internal Validity: One of the major contributions of our work is that we combine domain-specific features into duplicate bug report detection. There maybe different ways to combine the domain-specific features, and we only discuss one combination style. However, our integration of features into the neural network model after max-pooling layer has been validated to be more effective than baseline methods. Other integration styles will be discussed comprehensively in our future work.

VI. RELATED WORK

Our proposed approach is inspired by the research on duplicate bug detection.

Many previous studies have focused on detecting duplicate bug reports [2], [4]–[7], [13], [26]. Per Runeson et al. develop a tool aiming to employ the NLP techniques to help automate detection of defect reports. This paper presents the processing

TABLE VI
Performance Comparison with Different Word Embeddings

Approaches	DataSets	F-measure			Accuracy		
		Random	Glove	Word2vec	Random	Glove	Word2vec
Traditional CNN	Hadoop	0.642	**0.881**	0.853	0.714	0.869	**0.873**
	HDFS	0.815	0.792	**0.840**	0.823	0.702	**0.832**
	Spark	0.764	**0.833**	0.832	0.794	**0.855**	0.803
	MapReduce	0.693	0.725	**0.739**	0.753	0.769	**0.815**
DBR-CNN	Hadoop	0.749	0.917	**0.926**	0.723	**0.950**	0.945
	HDFS	**0.943**	0.909	0.870	0.904	**0.943**	0.873
	Spark	**0.964**	0.944	0.925	**0.951**	0.935	0.920
	MapReduce	0.894	**0.944**	0.892	0.874	0.919	**0.933**

steps of defect reports by using NLP in detail- tokenization, stemming, and stop words removal. And the experimental results show that about 40% of the marked duplicates could be found [2]. Wang et al. employ not only the word frequency but also the inverse document frequency. In addition, the execution information is also added in detecting process [4]. Jalbert et al. propose a novel system which automatically classifies duplicate bug reports as they arrive to save developer time. This system employs surface features, textual semantics, and graph clustering to detect duplicate bugs. Furthermore, they evaluate their approach by employing a dataset of 29,000 bug reports for Mozilla project, and the experimental results denote that their system is able to reduce development cost by filtering out 8% of duplicate bug reports [5]. Sun et al. present several approaches to address the duplicate bug reports [6], [7], [13], [26]. In [6], the discriminative models for information retrieval is leveraged to detect duplicate bug reports. In particular, Sun et al. improve the accuracy by extending the BM25F- which is an effective similarity formula in information retrieval community. And they also propose a new retrieval function REP for specific bug repositories. Yang et al. propose a novel approach which combines a traditional IR technique and a word embedding technique [27]. In addition, these factors also are taken in consideration that include bug titles, descriptions, bug product and component They calculate three similarity scores according to TF-IDF, word embedding vector, and bug product & component respectively. And then, the combined score of these three similarity scores is used to similar bug recommendation.

However, the performance of duplicate detection process should be enhanced. Different from above approaches, we leverage deep leaning technique to automate the detection process of duplicate bug reports, in which both syntactic and semantic features are extracted without manually choosing and searching of key words.

Hindle et al. provide empirical evidence supportive of code can be usefully modeled by statistical language models and such models can be leveraged to support software engineers by using n-gram model [28]. Recently, more attentions have been paid to deep learning techniques in duplicate questions [11], [12], [14]. Bogdanova et al. employ CNN to generate distributed vector representations for pairs of questions with a similarity metric [14]. The experimental results show that CNN with in-domain word embeddings achieves high performance. Homma et al. leverage a Siamese Gated Recurrent Unit (GRU) NN to encode each sentence to detecting duplicate questions [11]. What's more, they demonstrate data augmentation can be employed to enhance the performance of Siamese NN model. Addair explores three types of deep learning technologies for detecting duplicate question pairs, such as CNNs, long short-term memory networks and a hybrid model [12].

Different from these approaches, our approach employs deep leaning to produce CNN-based features of bug reports, and concatenates the CNN-based features and domain-specific features to further enhance the performance.

Budhiraja et al. propose an approach named Deep Word Embedding Network (DWEN) to compute bug report similarity by using the notion of word embeddings [29]. They employ the Mozilla Projects and Open Office Projects duplicate bug report datasets and compare IR-based method and Latent Dirichlet Allocation (LDA) approaches, and the experimental results show that the DWEN approach is able to perform better than IR-based approach and LDA approach. In addition, Budhiraja et al. combine LDA and word embeddings to take advantage of both for duplicate bug report detection [30]. They first use LDA model to calculate the similarity for a new bug report with all the existing bug reports, and top-n bug reports can be obtained. After that, top-K closest bug reports are shown to the Triager by using a word embedding model. their study on a real world dataset of Firefox project show that there is potential in combining both LDA and word embeddings for duplicate bug report detection. Deshmukh et al. employ siamese CNN and Long Short Term Memory (LSTM) to construct a retrieval and classification model for duplicate and similar bug detection and retrieval [31]. They use three different types of NN to concatenate the final encoding of bugs which our approach only employ the CNN to encode the bugs.

Our DBR-CNN approach differs from the above approaches, in which we improve traditional CNN model in combination with some domain-specific features extracted from bug reports. Furthermore, experimental analysis on bug reports from four different popular open-source projects demonstrates the application generalization of our approach.

VII. Conclusion

In this paper, we construct a CNN-based framework to automate the detection process of duplicate bug reports, which aims to exploit the DL in automated detecting to improve the detection performance. By employing DL technology and combining the domain-specific features, our approach takes advantage of both syntactic and semantic information that benefits the automated detecting of duplicate bug reports. Moreover, the experimental results on four different bug report datasets confirm that our approach improves the detection performance remarkably and our approach is extensible and flexible by combing multiple domain-specific features in different bug reports platforms.

To further enhance the performance, in the future, we will continue to optimize the CNN-based framework and enrich the experiments. Furthermore, more bug report datasets will be collected to exploit the new features by employing DL technologies.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant No. 61502401).

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. ACM, 2005, pp. 35–39.

[2] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007, pp. 499–510.

[3] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 361–370.

[4] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 2008, pp. 461–470.

[5] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. IEEE, 2008, pp. 52–61.

[6] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*.

[7] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 253–262.

[8] L. Hiew, "Assisted detection of duplicate bug reports," Ph.D. dissertation, University of British Columbia, 2006.

[9] L. Feng, L. Song, C. Sha, and X. Gong, "Practical duplicate bug reports detection in a large web-based development community," in *Asia-Pacific Web Conference*. Springer, 2013, pp. 709–720.

[10] S. Banerjee, Z. Syed, J. Helmick, M. Culp, K. Ryan, and B. Cukic, "Automated triaging of very large bug repositories," *Information and Software Technology*, vol. 89, pp. 1–13, 2017.

[11] Y. Homma, S. Sy, and C. Yeh, "Detecting duplicate questions with deep learning," in *Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016)*. IEEE, 2016, pp. 1–8.

[12] T. Addair, "Duplicate question pair detection with deep learning," *Stanf. Univ. J*, 2017.

[13] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012, pp. 70–79.

[14] D. Bogdanova, C. dos Santos, L. Barbosa, and B. Zadrozny, "Detecting semantically equivalent questions in online user forums," in *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, 2015, pp. 123–131.

[15] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[16] T. Mikolov, "Language modeling for speech recognition in czech," Ph.D. dissertation, Masters thesis, Brno University of Technology, 2007.

[17] T. Mikolov, J. Kopecky, L. Burget, O. Glembek *et al.*, "Neural network based language models for highly inflective languages," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. IEEE, 2009, pp. 4725–4728.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, pp. 1–12, 2013.

[19] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2017, pp. 318–328.

[20] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4277–4280.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[22] M. S. Rakha, C.-P. Bezemer, and A. E. Hassan, "Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports," *IEEE Transactions on Software Engineering*, 2017.

[23] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820*, 2015.

[24] W. Zhao, "Research on the deep learning of the small sample data based on transfer learning," in *AIP Conference Proceedings*, vol. 1864, no. 1. AIP Publishing, 2017, p. 020018.

[25] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[26] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. IEEE, 2012, pp. 385–390.

[27] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun, "Combining word embedding with information retrieval to recommend similar bug reports," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 127–137.

[28] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 837–847.

[29] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "Poster: Dwen: deep word embedding network for duplicate bug report detection in software repositories," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 193–194.

[30] A. Budhiraja, R. Reddy, and M. Shrivastava, "Poster: Lwe: Lda refined word embeddings for duplicate bug report detection," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 165–166.

[31] J. Deshmukh, S. Podder, S. Sengupta, N. Dubash *et al.*, "Towards accurate duplicate bug retrieval using deep learning techniques," in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 2017, pp. 115–124.