Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks

Jianjun He
Chongqing University
Chongqing, China
jjhe@cqu.edu.cn

Ling Xu*
Chongqing University
Chongqing, China
xuling@cqu.edu.cn

Meng Yan
Chongqing University
Chongqing, China
mengy@cqu.edu.cn

Xin Xia
Monash University
Melbourne, VIC, Australia
xin.xia@monash.edu

Yan Lei
Chongqing University
Chongqing, China
yanlei@cqu.edu.cn

ABSTRACT

Developers rely on bug reports to fix bugs. The bug reports are usually stored and managed in bug tracking systems. Due to the different expression habits, different reporters may use different expressions to describe the same bug in the bug tracking system. As a result, the bug tracking system often contains many duplicate bug reports. Automatically detecting these duplicate bug reports would save a large amount of effort for bug analysis. Prior studies have found that deep-learning technique is effective for duplicate bug report detection. Inspired by recent Natural Language Processing (NLP) research, in this paper, we propose a duplicate bug report detection approach based on Dual-Channel Convolutional Neural Networks (DC-CNN). We present a novel bug report pair representation, i.e., dual-channel matrix through concatenating two single-channel matrices representing bug reports. Such bug report pairs are fed to a CNN model to capture the correlated semantic relationships between bug reports. Then, our approach uses the association features to classify whether a pair of bug reports are duplicate or not. We evaluate our approach on three large datasets from three open-source projects, including Open Office, Eclipse, Net Beans and a larger combined dataset, and the accuracy of classification reaches 0.9429, 0.9685, 0.9534, 0.9552 respectively. Such performance outperforms the two state-of-the-art approaches which also use deep-learning techniques. The results indicate that our dual-channel matrix representation is effective for duplicate bug report detection.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools.*

KEYWORDS

Duplicate Bug Report Detection, Software Maintenance, Software Quality Assurance, Dual-Channel, Convolutional Neural Networks.

ACM Reference Format:
Jianjun He, Ling Xu, Meng Yan, Xin Xia, and Yan Lei. 2020. Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks. In *28th International Conference on Program Comprehension (ICPC '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3387904.3389263

1 INTRODUCTION

Modern software projects use bug tracking systems, such as Bugzilla to store and manage bug reports [13, 30]. These bug reports are submitted by developers, testers, and end-users to describe the software problems they encounter. Bug reports can help to guide software maintenance and repair activities [10, 16, 39]. Actually, with the development of software systems, several hundreds of bug reports are submitted to bug tracking systems each day [3]. Duplicate bug reports occur when more than one person submits bug reports for the same bug [21, 43]. Since bug reports are always written in natural language, the same bug may be described in many different ways. For example, Table 1 depicts an example of two duplicate bug reports **#4524** and **#9002** from the Open Office's bug tracking system. It is observed that there are few identical keywords in these two bug reports. Although they describe the same bug about the arrow placed too high with respect to the letters, the vocabularies they used are different.

With a huge amount of bug reports, detecting duplicate bug reports manually is a laborious process. It is impossible to provide a standard style of bug reports written in natural languages. Therefore, automatic duplicate bug report detection is a meaningful task that can help avoid fixing the same bug repeatedly [4, 7, 35, 44]. In recent years, many automatic duplicate bug report detection techniques have been proposed to address this issue. Some of these approaches, such as topic models [6, 14, 24, 38] focus on calculating the similarity based on the textual description. Some other approaches are adopted to mine potential features through self-learning algorithms, such as Learning to Rank (L2R) [25], Support Vector Machine (SVM) [40], etc. As shown in Table I, it is difficult for typical text similarity techniques to detect duplicate bug reports based on keywords. Recently, state-of-the-art deep-learning techniques have been proven superior to traditional approaches in

*Corresponding author

ICPC '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7958-8/20/05…$15.00
https://doi.org/10.1145/3387904.3389263

Table 1: An example of two duplicate bug reports.

Tag Name	Content	
Bug ID	4524	9002
Product	Math	Math
Component	UI	Code
Summary	space between a vector and its arrow too large.	formating of font attributes.
Description	Hi, The space between a vector and its arrow is too large making the formula too high. It doesn't matter much when the formula is a paragraph of its own but it looks clumsy when placed among the text of a paragraph. To make myself clear, copy this text in a sxw file then isert in the middle of the previous paragraph the formula "vec u" or the formula "**widevec** AB". Compare with what you'd get inserting the formula "overline AB". Thanks	The attibutes: hat, grave, tilde, check, bar, vector, and so on are too far removed from the font. Seems to be a problem with the font definitions used. Workaround are **widevec**, widehat, widebar etc. Unfortunatelly the 'wide' version does not exist for all attributes. Also,'bold' in formulae is tranlated into some sort of arial font with poor spacing wthin characters. It is unfortunate that this has changed from SOv5 which used the more conventional mathematical notation of Times bold for that, which incidentally has better character kerning.

Natural Language Processing (NLP) [26]. Deshmukh *et al.* [12] first propose a deep-learning approach to detect duplicate bug reports. They use Siamese Single Layered Neural Network to process structured information, Convolutional Neural Networks (CNN) [19] to process description and Long Short Term Memory (LSTM) [33] to process summary. The two bug reports are used as inputs for encoding separately, and then the two encodings are calculated to obtain similarity. Budhiraja *et al.* [8] extract the unstructured information (summary and description) in the bug report and use word embeddings [9] to convert each bug report into a matrix. They average each column of the matrix to obtain a vector of fixed dimensions to represent a bug report. Two vectors representing bug reports are concatenated to fed into deep neural networks to predict the similarity of the two bug reports.

In this paper, we propose a novel duplicate bug report detection approach by constructing a Dual-Channel Convolutional Neural Networks (DC-CNN) model. In our approach, each bug report is converted to a two-dimensional matrix by employing word embeddings. This matrix is similar to the grayscale image, so we call it a single-channel matrix. Instead of encoding the two reports separately like Deshmukh *et al.*[12], we combine the two single-channel matrices of two bug reports into a dual-channel matrix to represent a bug report pair.

The deep-learning techniques have been found to be effective for dealing with text classification [41]. Compared with other deep neural network models like Recurrent Neural Networks (RNN) [23] and Long Short Term Memory (LSTM), CNN can handle multi-channel input. Thus, we choose CNN to construct a classifier that aims to capture the relationship among the contextual words by using different sizes of kernels.

In order to verify the effectiveness of the proposed approach, we consider the following research questions:

RQ1: *Compared with the state-of-the-art deep-learning-based approaches, how effective is our DC-CNN?*

We compare our DC-CNN with other two state-of-the-art deep-learning-based duplicate bug report detection approaches. The results indicate that DC-CNN has better capability to classify duplicate bug reports and non-duplicate bug reports.

RQ2: *Compared with Single-Channel Convolutional Neural Networks based approach, how effective is our DC-CNN?*

To prove that the dual-channel CNN model is valid, we build a single-channel CNN model and compare the two on multiple measures. When combining two reports into a dual-channel report pair, CNN can better extract their associated features.

RQ3: *How effective is our approach when modeling without structured information?*

We process some structured information, including component and product, into unstructured forms and add them to other unstructured information, including summary and description, to help better classify. We observe the results of removing the structured information and find that processing structured information into unstructured forms does improve the accuracy of the classification, but summary and description are still dominant.

RQ4: *How effective is our approach in a cross-project detection setting?*

When a project lacks labeled data, an alternative solution is using labeled data from other project(s) as training project(s), i.e., the cross-project setting. For each project, we use the data of it as the target project and use the data from other project(s) as training project(s) to build a detection model. The experimental results reveal that when a project lacks labeled data, other project(s) can be used as training project(s) to build a detection model, but the appropriate project(s) need to be selected.

The main contributions of this paper are as follows:

(1) We propose a novel duplicate bug report detection approach using a dual-channel CNN model. This approach makes use of both structured and unstructured information in the bug report. The novelty of our approach is the use of dual-channel matrix for representing bug report pairs. Such pairs are fed to CNN to extract the association features between the two bug reports. Finally, our approach uses the association features to classify whether a pair of bug reports is duplicate or not.
(2) We compare our approach with two state-of-the-art deep-learning-based approaches proposed by *Deshmukh et al.* (Siamese Networks) [12] and Budhiraja *et al.* (DWEN) [8] on three large open-source projects (Open Office, Eclipse, and Net Beans) [18] and a larger dataset combined of the three, containing a total of 531,267 bug report pairs. The experimental results show that our approach outperforms

the state-of-the-art deep-learning-based approaches for duplicate bug report detection.

This paper is organized as follows. Section 2 presents our approach in detail. In section 3, we describe the datasets used and the evaluation metrics adopted. Section 4 presents the experimental results and analysis. In Section 5, we discuss the possible reason for why our DC-CNN works for duplicate bug report detection. Section 6 presents the related work. Section 7 summarizes the main threats of our study. The conclusion and future work are presented in Section 8.

2 APPROACH

Figure 1 presents the overall framework of DC-CNN which contains three phases: data preparation, training, and deployment. In the data preparation phase, the useful fields including component, product, summary, and description are extracted from bug reports. For each bug report, both the structured and unstructured information is placed into a text document together. After preprocessing, texts of all the bug reports are collected and formed into a corpus. A word2vec model [22, 28] is used to capture semantic regularities on the corpus. We convert each bug report represented by text into a single-channel matrix. To investigate the relationship between bug reports, we combine the reports of single-channel matrices into bug report pairs represented by dual-channel matrices. Then we take a part of them as the training set and another part of them as the testing set. In the training phase, we take the training set as input to train the CNN model. In the deployment phase, the testing set is fed to the trained model to predict the similarity of each bug report pair which is an independent probability. The similarity is then compared with the set threshold to classify a pair of bug reports as duplicate or not.

2.1 Data Preparation

2.1.1 Data Extraction. Table 2 shows a sample of a bug report in Eclipse. The bug report consists of structured information such as product, component, priority, version, etc. and unstructured information such as summary and description. Structured information is usually an optional attribute, while unstructured information is the text description of the bug. In our experiments, we only consider product, component, summary, and description and put them into a single text file for each bug report. For the bug report **#11112** shown in Table 2, after data extraction, we get a piece of text containing both structured and unstructured information: *"**Product:** Platform. **Component:** SWT. **Summary:** Multi monitors not correctly supported. **Description:** Situation: Windows XP with two monitors. If you run eclipse on your secondary monitor, code completion and comments are displayed on the primary monitor."*.

2.1.2 Preprocessing. For each bug report, we perform typical preprocessing steps including tokenization, stemming, stop words removal and case conversion. We do the preprocessing using Lucene's StandardAnalyzer which is a full-text search engine toolkit for Apache [5]. When removing stop words, we use a standard vocabulary of English stop words. We find that even in two completely unrelated reports, there are also some identical words. These words are usually very professional words, including *java*, *com*, *org*. Due to frequent occurrences, we also add them to the stop word list. After the above processing, there are still some isolated numbers which are meaningless and non-English letters such as Japanese left in the text, and we remove them.

Table 2: A sample of a bug report.

Tag Name	Content
Bug ID	11112
Product	Platform
Component	SWT
Summary	Multi monitors not correctly supported
Status	RESOLVED
Resolution	DUPLICATE
Duplicates	7232
Priority	P3
Severity	major
Version	2.0
Created	2002-03-11 13:53:00 -0500
Modified	2002-05-15 12:07:18 -0400
Description	Situation: Windows XP with two monitors. If you run eclipse on your secondary monitor, code completion and comments are displayed on the primary monitor.

2.1.3 Dual-Channel Matrix Presentation. After preprocessing, all the words in all the bug reports are made up of a corpus. We use word2vec with the Continuous Bag of Words (CBOW) model on the corpus to obtain the word vector of each word and the word vector dimension is set to 20. Then, we can get a two-dimensional matrix of each bug report. When a new bug report contains a word that is not in the corpus, set the word to a 0 vector. Since the input of CNN is required to be a fixed shape and the length of the bug report is different, we adopt a truncation strategy. Parts of more than 300 words will be discarded, and when the text contains less than 300 words, it will be supplemented with zero vector. Similar to the grayscale image, we can also call this two-dimensional matrix single-channel matrix.

The siamese networks used by Deshmukh *et al.* [12] encode the two bug reports separately. Then, the encodings of the two bug reports are operated to obtain the similarity of them. Our approach aims to find a new bug report pair representation to avoid encoding the two bug reports separately. In the image field, a grayscale image is a single-channel matrix, while a color image is a multi-channel matrix which can be seen as a combination of multiple single-channel matrices. Inspired by this, we try to combine the two single-channel matrices into a dual-channel matrix to represent a bug report pair and then label it with duplicate (1) or non-duplicate (0). This process is shown in Figure 2.

Compared with single-channel, there are some benefits to using the dual-channel presentation of bug report pair. Firstly, the two reports can be processed jointly by CNN. Therefore, the training speed is improved. Secondly, it is found that the use of dual-channel data to train the CNN model achieves a high accuracy [20]. For a dual-channel CNN architecture, it can capture the correlated

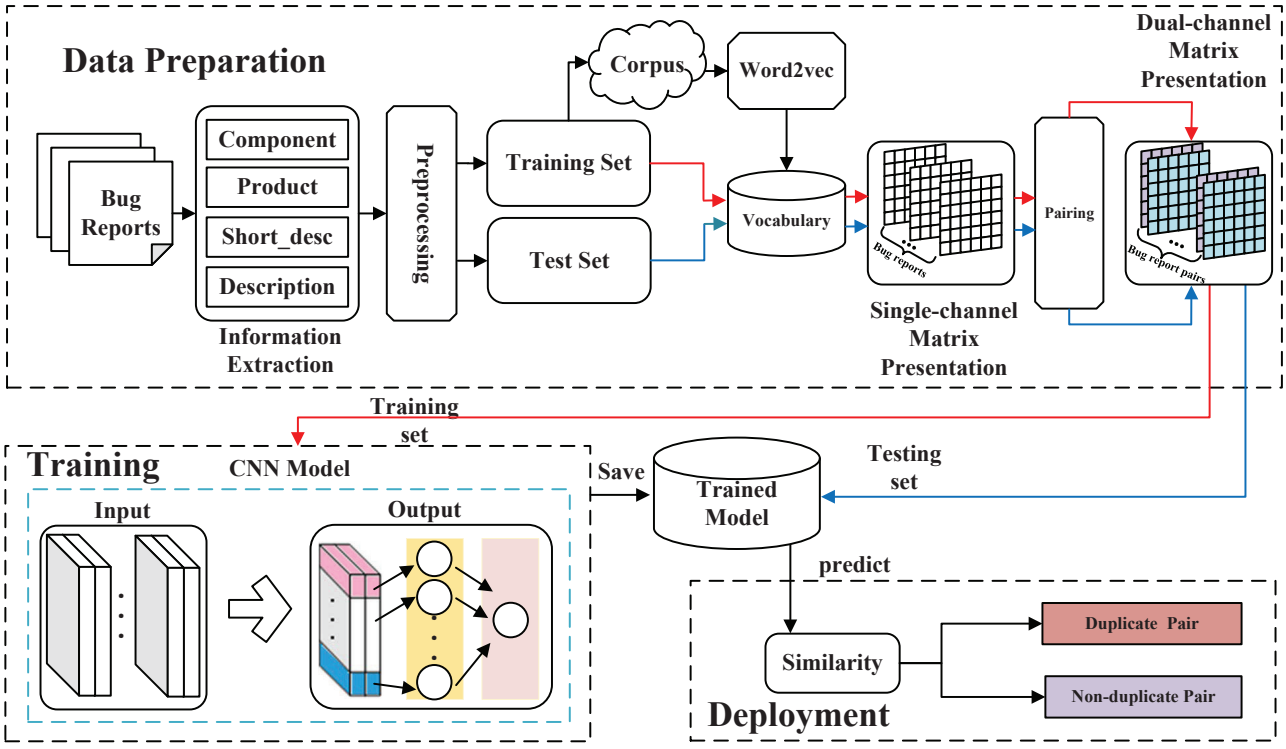

Figure 1: Dual-Channel convolutional neural networks model.

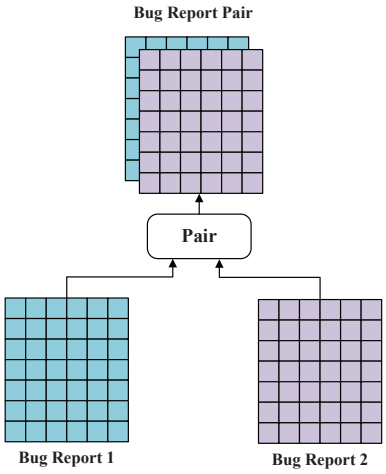

Figure 2: Combining two bug reports into a bug report pair.

semantic relationships between bug reports by convolving the input which is a concatenation of two single-channel matrices converted from bug reports.

2.2 CNN Model

In order to extract features from a bug report pair, we set three different sizes of kernels in each convolutional layer. Thus, there are three branches in the first convolutional layer. For each of the three branches, there still exist three new branches in the second convolutional layer. Since the three branches have a highly similar architecture, Figure 3 presents the overall workflow of the CNN model with only a branch of the first convolutional layer in detail. All of the convolution operations use Conv2D. Table 3 provides the detail parameter settings about our whole CNN model. For the hyperparameters, the learning rate is set to 0.001, the batch_size is 64, and *Adam* is used as the optimizer. In addition, we adopt the early stopping strategy. The model will stop training when the verification loss no longer falls within 5 epochs.

In the first convolutional layer, there are n_{k_1} kernels $k_1 \in \mathbb{R}^{d \times k_w}$, where d is the length of kernels and k_w is the width of kernels. Since each row of the input matrix represents a word, these kernels' widths k_w are identical and equal to m which denotes the dimension of word vector. After the first convolution, two channels of a dual-channel matrix are merged into one channel. Thus, we treat these two reports as a whole to extract features. Given the length of input l ($l = n_w$), the padding P ($P = 0$) and the stride S ($S = 1$), the length of the output O_1 can be calculated as:

$$O_1 = \frac{l - d + 2P}{S} + 1 \tag{1}$$

The output of the first convolutional layer is in the shape of $O_1 \times 1 \times n_{k_1}$. In order to further extract the association features between the two reports, we reshape it into $O_1 \times n_{k_1} \times 1$ and convolve it once again. In the second convolutional layer, we set three different kinds of kernels $k_2 \in \mathbb{R}^{d \times k_w}$ ($d = 1, 2, 3$ in Table 3, $k_w = n_{k_1}$) and the number of each type of the kernel is n_{k_2}. After this

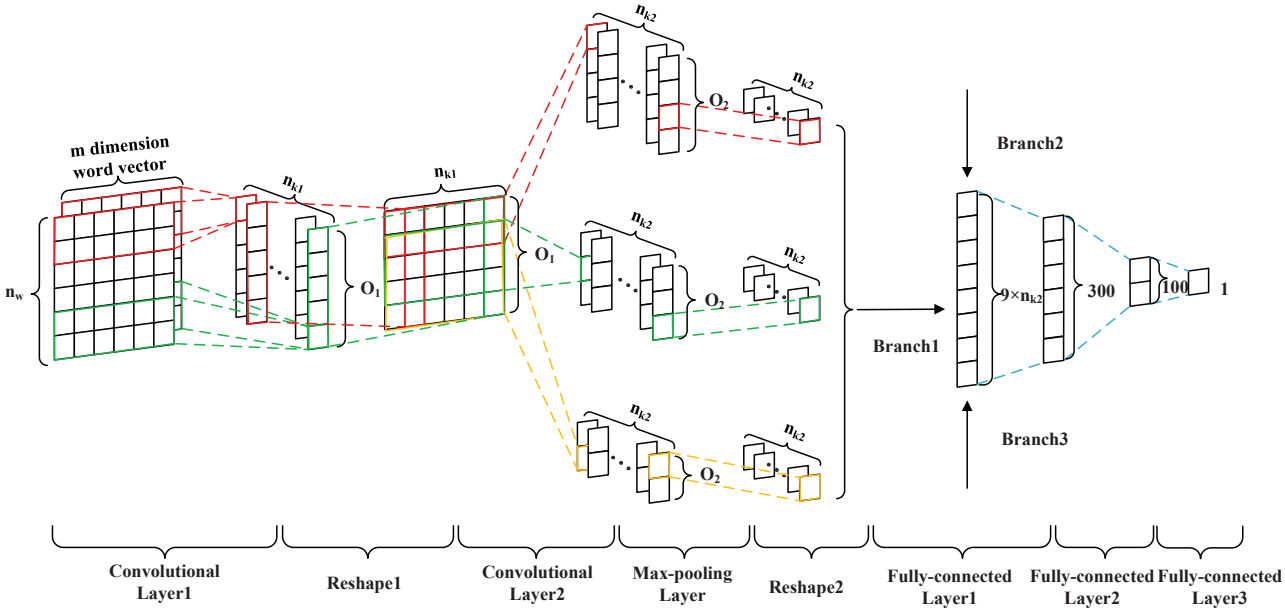

Figure 3: The overall workflow of CNN model.

Table 3: The parameter settings for CNN model.

Parameter Name	Convolution 1			Convolution 2			Max-pooling	FC 1	FC 2	FC 3
	Branch 1	Branch 2	Branch 3	Branch 1	Branch 2	Branch 3				
Kernel Size ($d \times k_w$)	1×20	2×20	3×20	1×100	2×100	3×100	-	-	-	-
Kernel Number	100	100	100	200	200	200	-	-	-	-
Output Dimension	-	-	-	-	-	-	-	300	100	1
Activation Function	Relu	Relu	Relu	Relu	Relu	Relu	Relu	Relu	Relu	Sigmoid

convolution, we get three kinds of feature maps with the size of $O_2 \times 1 \times n_{k_2}$, where O_2 can also be calculated as Formula 1 according to l ($l = O_1$) and different kernels' length d. In order to preserve the most important features extracted by convolution and reduce the dimensions of the data, we use O_2-max pooling for all the feature maps. Thus, each feature map is subsampled into the shape of $1 \times 1 \times n_{k_2}$. Finally, we reshape and concatenate all the feature maps from all the three branches of the first convolutional layer into a $9 \times n_{k_2}$ dimensional vector as the input to the fully-connected layer. After three fully-connected layers, we end up with an independent probability $sim_{predict}$ which represents the predicted similarity between the two reports.

At each layer except the last fully-connected layer, we use the *Rectified Linear Units* (*Relu*) as the activation function to extract more nonlinear features. In the last layer, in order to get a value between 0 and 1 to represent $sim_{predict}$, we use *Sigmoid* instead. Given the output of the second fully-connected layer $T = [x_1, x_2, ..., x_{100}]$ and weight vector $W = [w_1, w_2, ..., w_{100}]$, $sim_{predict}$ can be calculated as:

$$sim_{predict} = \frac{1}{1 + e^{-(\sum_{i=1}^{100} w_i x_i + b)}}, \tag{2}$$

where i is the i_{th} element of vectors and b refers to the bias.

In order for the model to automatically adjust parameters, along with *Sigmoid*, our loss function is set to:

$$loss = -\sum_{i=1}^{n} sim_{predict\,i} log(label_{real\,i}) + (1 - sim_{predict\,i}) log(1 - sim_{predict\,i}), \tag{3}$$

where $label_{real}$ denotes the actual label of a bug report pair, i represents the i_{th} pair and n denotes the total number of pairs.

3 STUDY SETUP

In this section, we give the details of our experimental setup. We first describe the datasets used in our experiments. Then the evaluation metrics adopted are introduced.

3.1 Datasets

We use public datasets which are collected and processed by Lazar [18]. The datasets contain three large open-source projects: Open Office, Eclipse and Net Beans. Open Office is an office software similar to Microsoft Office. Eclipse and Net Beans are open-source integrated development environments. In order to make experiments with more training samples, we create a larger dataset by

merging these three datasets in an additive way and name it "Combined". The datasets also provide us with bug report pairs, part of which in Open Office are shown in Table 4. When the **RESOLUTION** of a bug report is **DUPLICATE**, this bug report will be combined with its corresponding duplicate one to form a duplicate pair. In our case, a non-duplicate pair refers to two bugs without duplicates. This approach is adopted to generate random samples of non-duplicates.

Table 4: Examples of bug report pairs.

Bug ID 1	Bug ID 2	Relationship
24359	53249	duplicate
66799	66919	duplicate
1635	93577	non_duplicate
49387	116949	non_duplicate

Table 5: The complete datasets.

Dataset	Duplicate	Non_duplicate
Open Office	57340	41751
Eclipse	86385	160917
Net Beans	95066	89988
Combined	238791	292476

In the analysis of all the pairs in each dataset, we notice that there are some problems in these pairs. Firstly, some pairs are duplicates. For example, the pair (#200622, #197347, *duplicate*) in Open Office appears five times. Secondly, some pairs represent the same association, such as the pairs (#159435, #164827, *duplicate*) and (#164827, #159435, *duplicate*) in Eclipse. Thus, we remove these items. Table 5 shows the number of pairs in the datasets we finally get.

3.2 Evaluation Metrics

The output of our CNN model is a value ranging from 0 and 1, which represents the similarity between two bug reports in a pair. For further classification, a threshold (0.5) is set. When we have got $sim_{predict}$ (recall Section 2), we can calculate $label_{predict}$ which means the predicted label of a pair as follow:

$$label_{predict} = \begin{cases} 1, & sim_{predict} \geq 0.5 \\ 0, & sim_{predict} < 0.5 \end{cases} . \quad (4)$$

Bug report pairs can be divided into four categories based on $label_{predict}$ and $label_{real}$: **TP**, **TN**, **FP**, **FN**.

TP indicates the number of pairs that are correctly predicted as duplicates, **TN** indicates the number of pairs that are correctly predicted as non-duplicates, **FP** indicates the number of pairs that are incorrectly predicted as duplicates, and **FN** indicates the number of pairs that are incorrectly predicted as non-duplicates. These four basic performance evaluation metrics are used to the following evaluation methods.

Accuracy indicates the proportion of all correctly predicted bug report pairs to all pairs, and it shows the performance of our model to classify all the bug report pairs correctly. *Accuracy* is calculated as $\frac{TP+TN}{TP+TN+FP+FN}$.

Recall indicates the proportion of all bug report pairs that are correctly predicted as duplicates to all actual duplicate pairs. *Recall* is calculated as $\frac{TP}{TP+FN}$.

Precision indicates the proportion of all bug report pairs that are correctly predicted as duplicates to all pairs that are predicted as duplicate. *Precision* is calculated as $\frac{TP}{TP+FP}$.

F1-Score is a comprehensive evaluation of *Recall* and *Precision*, which is the harmonic mean of them. It evaluates if an increase in *Precision* (or *Recall*) outweighs a reduction in *Recall* (or *Precision*), respectively and provides a balanced view of *Precision* and *Recall*. *F1-Score* is calculated as $\frac{2\times Recall\times Precision}{Recall+Precision}$.

AUC is the Area Under the Curve of Receiver Operator Characteristic (*ROC*). *ROC* can be plotted by computing the True Positive Rates (*TPR*) and the False Positive Rates (*FPR*) according to different thresholds. *TPR* is calculated as $\frac{TP}{TP+FN}$ and *FPR* is canlculated as $\frac{FP}{FP+TN}$. Taking all the *FPR* values as the horizontal axis and all the *TPR* values as the vertical axis, the *ROC* curve can be obtained. Already mentioned above, when determining whether a pair of bug reports is duplicate, the model first outputs a likelihood socre for this pair to be duplicate. Then, the likelihood score needs to be compared with a threshold. If the likelihood score is higher than the threshold, this pair of bug reports are duplicate, otherwise this pair of bug reports are non-duplicate. The threshold can range from 0 to 1. *AUC* measures the performance of approaches across all the thresholds. Thus, it is a threshold independent measure.

Normalized Improvement indicates the room for improvement, which is proposed by Costa *et al.* [11] to avoid inflated results caused by directly comparing difference or direct proportion of results. Since prior studies have already achieved a high performance (e.g., $Accuracy > 0.8$), the *Normalized Improvement* is more appropriate for measuring the improvement over baselines in our context by following Costa *et al.* [11]. Formally, *Normalized Improvement* is calculated as $\frac{Per_{ours}-Per_{base}}{1-Per_{base}}$, where Per_{ours} is the performance of our approach and Per_{base} is the performance of the baseline.

To evaluate the effectiveness of our approach, we adopt a stratified 5-fold cross-validation setting. In detail, we randomly divide the dataset into five folds by using stratified random sampling. Such sampling technique aims to keep the ratio of duplicate report pairs and non-duplicate report pairs in each fold to be the same as the original dataset. Then, four folds are used to train the model, while the remaining one fold is used to evaluate the performance of our approach. We repeat this process five times, so that each fold is used once as the testing set. As a result, there are five effectiveness values and we report the average values for each performance measures.

4 EXPERIMENTAL RESULTS

In this section, we focus on answering the following four research questions to demonstrate the experimental results and their implications.

4.1 RQ1: Compared with the state-of-the-art deep-earning-based approaches, how effective is our DC-CNN?

Motivation. To the best of our knowledge, the state-of-the-art deep-learning-based approaches for duplicate bug report detection

Table 6: The results of our approach compared with other approaches.

Dataset	Accuracy			Recall			Precision			F1-Score			AUC		
	Siamese	DWEN	DC-CNN	Siamese	DWEN	DC-CNN	Siamese	DWEN	DC-CNN	Siamese	DWEN	DC-CNN	Siamese	DWEN	DC-CNN
Open Office	0.8399	0.9304	**0.9429**	0.8586	0.9409	**0.9670**	0.8638	0.9389	**0.9365**	0.8612	0.9399	0.9515	0.9223	0.9780	**0.9843**
Eclipse	0.8625	0.9436	**0.9685**	0.7535	0.9079	**0.9409**	0.8367	0.9267	**0.9680**	0.7929	0.9172	0.9543	0.9836	0.9836	**0.9950**
Net Beans	0.8085	0.9221	**0.9534**	0.7282	0.9070	**0.9513**	0.8782	0.9394	**0.9576**	0.7962	0.9229	0.9544	0.9005	0.9774	**0.9893**
Combined	0.8275	0.9325	**0.9552**	0.7958	0.9309	**0.9576**	0.8157	0.9199	**0.9435**	0.8056	0.9254	0.9505	0.9127	0.9809	**0.9906**

Table 7: The confusion matrix of the three approaches.

Dataset	TP			FP			TN			FN		
	Siamese	DWEN	DC-CNN	Siamese	DWEN	DC-CNN	Siamese	DWEN	DC-CNN	Siamese	DWEN	DC-CNN
Open Office	**9846**	**10789**	**11089**	1552	702	752	**6800**	**7650**	**7600**	1621	678	378
Eclipse	**13018**	**15685**	**16255**	2541	1240	536	**29644**	**30945**	**31649**	4258	1591	1021
Net Beans	**13846**	**17244**	**18087**	1920	1113	800	**16079**	**16886**	**17199**	5167	1769	926
Combined	**38003**	**44455**	**45731**	8584	3871	2740	**49948**	**54661**	**55792**	9753	3301	2025

are proposed by Deshmukh *et al.* (Siamese Networks) [12] and Budhiraja *et al.* (DWEN) [8]. The goal of our study is to propose a more effective deep-learning-based approach. Therefore, we compare DC-CNN with Deshmukh *et al.*'s and Budhiraja *et al.*'s approaches based on the same datasets. Since we do not have the source code of these two papers, we implement their approaches as described in their papers on our datasets. Deshmukh *et al.*'s approach includes two models - the retrieval model and the classification model. Since the classification model has the same task as our model, we implement it.

Results. Table 6 shows the results of Deshmukh *et al.*'s approach (Siamese Networks), Budhiraja *et al.*'s approach (DWEN) and our approach (DC-CNN). We focus on *Accuracy*, *F1-Score* and *AUC* to evaluate the performance of our approach. Compared with Siamese Networks, the *Normalized Improvement* in terms of *Accuracy* is 64.33%, 77.09%, 75.66%, and 74.03%, respectively. The *Normalized Improvement* in terms of *Accuracy* is 17.96%, 44.15%, 40.18%, and 33.63%, respectively, compared with DWEN. Compared with Siamese Networks, the *Normalized Improvement* in terms of *F1-Score* is 65.06%, 77.93%, 77.62%, and 74.54%, respectively. The *Normalized Improvement* in terms of *F1-Score* is 29.56%, 44.81%, 69.08%, and 33.65%, respectively, compared with DWEN. Compared with Siamese Networks, the *Normalized Improvement* in terms of *AUC* is 79.79%, 69.51%, 89.25%, and 89.23%, respectively. The *Normalized Improvement* in terms of *AUC* is 28.64%, 69.51%, 54.65%, and 50.78%, respectively, compared with DWEN.

Implications. According to Table 6, the performance of DC-CNN is better than that of Siamese Networks and DWEN on all four datasets. *Accuracy* is affected by changes of *TP* and *TN*, so we want to explore further the cause of the change from the confusion matrix which is shown in Table 7. It can be seen that for all the datasets, both *TP* and *TN* in our DC-CNN are on the rise (except for a slight decrease for *TN* in Open Office). Therefore, the improvement in terms of *Accuracy* is not only simply due to correctly classify more duplicate bug report pairs or only simply due to correctly classify more non-duplicate bug report pairs. Whether it's a duplicate bug report pair or a non-duplicate bug report pair, our DC-CNN has better classification performance. This further demonstrates that our DC-CNN model is excellent in duplicate bug report pair classifications.

4.2 RQ2: Compared with Single-Channel Convolutional Neural Networks based approach, how effective is our DC-CNN?

Motivation. To prove our dual-channel input of the bug report pair is effective, we also use the single-channel input of the bug report as our baseline. The dual-channel input in the DC-CNN is split according to the channel to regain two single-channel matrices representing bug reports. Then we keep the structure of CNN (part except for the fully-connected layers in Figure 3) unchanged and use it to extract features of two reports represented by single-channel matrices in a pair separately. The feature vectors of the two single-channel matrices encoded by CNN are calculated to obtain their cosine similarity. We call this approach Single-Channel Convolutional Neural Networks (SC-CNN).

Results. We evaluate their performances through *Accuracy*, *Recall*, *Precision*, *F1-Score* and the results are shown in Table 8, where the best values are highlighted in bold. It is observed that DC-CNN achieves better performance than SC-CNN with all these four metrics on all the datasets. Compared with SC-CNN, the *Normalized Improvement* in terms of *Accuracy* is 32.74%, 45.31%, 22.55%, 34.21%, the *Normalized Improvement* in terms of *Recall* is 45.27%, 7.94%, 23.43%, 42.78%, the *Normalized Improvement* in terms of *Precision* is 24.67%, 67.11%, 22.06% 26.91%, the *Normalized Improvement* in terms of *F1-Score* is 33.10%, 43.58%, 22.84%, 34.61%, and the *Normalized Improvement* in terms of *AUC* is 52.28%, 68.94%, 32.28%, 52.28% on Open Office, Eclipse, Net Beans, and Combined respectively.

Implications. All of these results indicate that the dual-channel CNN model is more effective compared with the approach using single-channel. For SC-CNN, every single bug report is transformed into a matrix and then fed into CNN to extract their features which are represented by feature vectors. Duplicate bug reports are detected by computing the similarity of two feature vectors. For DC-CNN, the matrices of two bug reports are concatenated into a dual-channel matrix and fed to CNN, then the matrices of two bug reports are convolved together, which is able to extract the deeper semantic relationship between the two reports and makes full use of the ability of CNN to capture local features.

Table 8: DC-CNN vs. SC-CNN using metrics Accuracy, Recall, Precision, F1-Score and AUC.

Dataset	Accuracy		Recall		Precision		F1-Score		AUC	
	SC-CNN	DC-CNN	SC-CNN	DC-CNN	SC-CNN	DC-CNN	SC-CNN	DC-CNN	SC-CNN	DC-CNN
Open Office	0.9151	**0.9429**	0.9397	**0.9670**	0.9157	**0.9365**	0.9275	**0.9515**	0.9671	**0.9843**
Eclipse	0.9424	**0.9685**	0.9358	**0.9409**	0.9027	**0.9680**	0.9190	**0.9543**	0.9839	**0.9950**
Net Beans	0.9397	**0.9533**	0.9364	**0.9513**	0.9456	**0.9576**	0.9409	**0.9544**	0.9842	**0.9893**
Combined	0.9319	**0.9552**	0.9259	**0.9576**	0.9227	**0.9435**	0.9243	**0.9505**	0.9803	**0.9906**

Table 9: The performance in cross-project setting.

Target Project	Training Project(s)	TP	FP	FN	TN	Accuracy	Recall	Precision	F1-Score
Open Office	Eclipse	200	2	11267	8350	0.4313	0.0174	0.9901	0.0342
	Net Beans	10976	6230	491	2122	0.6608	0.9571	0.6379	**0.7656**
	Eclipse + Net Beans	4740	343	6727	8009	0.6432	0.4133	0.9325	0.5728
Eclipse	Open Office	15308	23932	1968	8253	0.4763	0.8861	0.3901	0.5417
	Net Beans	4806	788	12470	31397	0.7319	0.2782	0.8591	0.4203
	Open Office + Net Beans	16845	20622	431	11563	0.5743	0.9750	0.4496	**0.6154**
Net Beans	Open Office	17757	16502	1256	1497	0.5202	0.9339	0.5183	**0.6666**
	Eclipse	2051	51	16962	17948	0.5403	0.1079	0.9757	0.1943
	Open Office + Eclipse	8821	1284	10192	16715	0.6899	0.4639	0.8729	0.6059

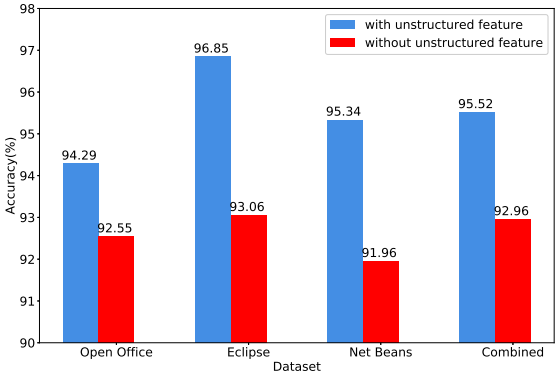

Figure 4: The effect of unstructured information.

4.3 RQ3: How effective is our approach when modeling without structured information?

Motivation. Structured information such as product, component, version, etc. provides useful information to distinguish two bug reports are duplicate or not. Many approaches always treat structured information as a single feature to improve the *Accuracy* of duplicate bug report detection. Unstructured information is the natural language description of the bug. For duplicate bug report detection, the CNN model is mainly used to process unstructured text since it has good performance in processing long text. Different from the previous methods, in this paper, we consider both structured and unstructured information as text data and put them together into a text document. The CNN model is applied to extract its features. In order to answer **RQ3**, we remove the structured information from the input and conduct comparative experiments without changing other conditions.

Results. It is observed from Figure 4, after removing the structured information, the *Accuracy* on all the datasets decreased (directly comparing the difference) by 1.74%, 3.79%, 3.38%, 2.56% on Open Office, Eclipse, Net Beans, and Combined respectively.

Implications. Experimental results show that it is useful to feed structured information along with unstructured information to CNN. We notice that after removing structured information, although the *Accuracy* has declined, this decline is not fatal. The reason might be that structured information only accounts for a small fraction of the whole text. The main part for CNN to extract features is still unstructured information.

4.4 RQ4: How effective is our approach in a cross-project detection setting?

Motivation. By default, we train our detection model by learning from the historical labeled data within the project. However, for projects with limited development history, there is often not enough labeled data for building a deep-learning-based model. An alternative solution is to build a detection model by learning from other projects, i.e., the cross-project setting as stated by the previous study [12]. Thus, in this section, we investigate the effectiveness of the cross-project setting when using our approach. In the cross-project setting, we use labeled data from other projects to train a model. Then, such model is used to detect duplicate bug reports for the target project.

In detail, for each target project, we construct three models using the data from the other two projects respectively and the data combined by the other two projects. Then, we evaluate the effectiveness of the three models by testing on the target project.

Results. Table 9 shows the evaluation results in the cross-project setting. We focus on the *F1-Score* since it can show the performance of the classification model from a more comprehensive perspective. For Open Office, the highest *F1-Score* is 0.7656. Such performance

Table 10: The summary and description of a duplicate pair.

Bug ID	Content	
25791	Summary	paper size bigger/ longer than A0 (1019x1019mm)
	Description	Trying to print a spreadsheet long! (2500mm) I have a plotter which is able to do so and able to receive postscript files. - choose format, page - page, format: user - it's impossible to choose a size bigger than 1019x1019mm (nevertheless which.
50245	Suamary	width / height of a drawing-sheet more than 119 cm
	Description	the maximum height / width of a drawing sheet is limitated on 119 cm x 119 cm. (This limitation is in Impress too.) Is it possible to set this limitation higher (e.g. unlimited)?

is achieved when learning from Net Beans project. For Eclipse, the highest *F1-Score* is 0.6154. Such performance is achieved when learning from the combined dataset of Open Office and Net Beans. For Net Beans, the highest *F1-Score* is 0.6666. Such performance is achieved when learning from Open Office.

Implications. From the experimental results, we can find that the cross-project setting is also doable when there is not enough labeled data in the target project. However, one remaining issue is how to select the proper training project(s). It's worth noting that when using Open Office as the training project, the evaluation result of Net Beans is better. When using Net Beans as the training project, the evaluation result of Open Office achieves a better performance. The reason might be that Open Office and Net Beans are projects from the same organization (*Sun*). Projects developed by the same organization may have a lot of similarities. Thus, when a target project lacks labeled data, an alternative solution is to build a cross-project model by earning from anther similar project when using our approach.

5 DISCUSSION: WHY DOES DC-CNN WORK?

In this section, we analyze the reason for why our DC-CNN works for duplicate bug report detection.

Our DC-CNN can be divided into two parts: Dual-Channel and CNN. We explain why CNN works first. Our duplicate bug report detection is actually a text classification problem. The key to text classification is to refine the central idea of a document or sentence accurately. The way to refine the central idea is to extract the keywords in the document or sentence as features. Convolution and pooling in CNN is such a feature extraction process. Since our input is a document, and the adjacent words in it are highly correlated, when we set up convolution kernels of different sizes, we not only consider the meaning of the words but also consider the word order and context. So, we can use CNN to train our classifier.

Dual-channel means that the convolution of CNN can process two sets of data once. In the field of Computer Vision (CV), pictures with only gray values have only one channel. If it is a color picture, there will be three images of RGB (i.e., three channels). We apply the idea of CV to the text, that is, let CNN extract the features of the two bug reports at a time and after the first convolution, the features of the two bug reports are merged.

Table 10 shows the summary and description of a duplicate pair on Open Office that were successfully detected by DC-CNN. Both the bug reports describe a problem with limited page size. When describing the page size, **#25791** uses *"size"* while **#50245** uses *"width/height"*. When making a request for a larger page selection, **#25791** uses *"bigger/longer"* while **#50245** uses *"higher"* and *"unlimited"*. At the first convolution, CNN extracts the features from both reports and fuses them. In the subsequent convolution and pooling, the features of this fusion are further refined. Finally, our DC-CNN model extracts the deep association features of this pair. Such a process makes it possible for even two reports that use different keywords to describe the same bug to be detected by our model.

6 RELATED WORK

In the literature, many approaches have been proposed for automatic duplicate bug report detection. In general, we can roughly divide these approaches into two categories: text similarity and machine learning.

Text similarity is the commonly used technique for duplicate bug report detection by computing textual similarity between two bug reports. Hiew *et al.* [15] propose a model using Vector Space Model (VSM) which calculates a report as a vector with Term Frequency-Inverse Document Frequency (TF-IDF) term weighting scheme. Based on VSM, Runeson *et al.* [29] first use Natural Language Process (NLP) to detect duplicate bug reports. Wang *et al.* [37] believe that only considering natural language information may not be sufficient for solving this problem. Hence, they use the execution information as a feature to perform duplicate bug report detection. However, only a small percentage of reports contain execution information and there is a significant limitation about this way. Sun *et al.* [31] propose REP which not only uses summary and description to calculate text similarity but also uses structured information including product, component, version and so on. To get a more accurate text similarity, they extend BM25F which is an effective similarity method. In addition to text similarity and structured similarity, Alipour *et al.* [2] also study the impact of contextual information on duplicate bug report detection. They apply LDA on these features and get a better result. The IR-based approaches perform well in both accuracy and time efficiency, but when a problem is described by different descriptive terms, the effect is unsatisfied. So Nguyen *et al.* [27] propose DBTM, which leverages both IR-based features and topic-based features. What's more, Sureka *et al.* [34] also use a character N-gram-based model

to detect duplicate bug reports. Aggarwal *et al.* [1] propose a simpler software-literature context approach to detect duplicate bug reports. This method reduces the manual effort used in contextual bug deduplication with only a little loss in terms of accuracy.

Machine learning is also a line of approach for the automatic detection of duplicate bug reports. Support Vector Machine (SVM) is a very classic method in machine learning. Jalbert *et al.* [17] use it to develop a classifier system which can filter out the duplicate bug reports. At the same time, they think that the previous approaches do not take into account the various features available in the report, so they use surface features, textual semantics and graph clustering on this model. Based on the work of Jalbert, Tian *et al.* [36] consider some new features and construct a linear model. Starting with directions of features and addressing imbalanced data, they improve accuracy. Sun *et al.* [32] use SVM to construct a discrimination model. They first divide the bug reports into two classes - duplicate and non-duplicate. Learning to Rank (L2R) is another useful machine learning approach. Based on L2R, Zhou *et al.* [42] consider textual and statistical features and use a stochastic gradient descent algorithm for them. The proposed approach works better than traditional IR approached like VSM and BM25F. With the application of word embedding in NLP, more and more researchers are using it to detect duplicate reports. Budhiraja *et al.* [9] use word embedding to convert bug reports into vectors and then calculate their similarity. Experimental results show that this approach has the potential to improve duplicate bug report detection.

Previous research has proposed some features to represent bug reports. However, few methods can extract the semantic relationship between bug reports. Deep-learning techniques perform excellently in the area of NLP tasks and can help to bridge this gap. Deshmukh *et al.* [12] first use deep-learning to perform duplicate bug report detection. They propose a retrieval model and a classification model based on Siamese Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM). Their experiments show that deep-learning makes great progress for this problem. Budhiraja *et al.* [8] use word2vec to convert each report into a matrix representation, and then average each column of the matrix, thus obtaining a vector of fixed dimensions to represent a bug report. They also concatenate the two vectors of a pair of bug reports and put them into a deep neural network to predict their similarity. The result of this approach also exceeds most of the previous duplicate bug report detection approach.

7 THREATS TO VALIDITY

In this section, we analyze several potential aspects that may threaten the validity of our approach and experiments.

Internal Validity. Threats to internal validity relate to potential errors in our experimental implementation. To reduce errors in our code, we have double-checked and fully tested our code, still there could be errors that we did not notice. Additionally, we implement our approach on top of the commonly used tools (e.g., Lucene for text preprocessing and Keras for CNN) to mitigate this issue.

In the implementation of other methods, although we strictly follow the description of their papers, there are some experimental details not mentioned in the papers. To address this issue, we set them to default.

External Validity. The threat to external validity is the generalizability of our experiment results. We have evaluated our approach on three large datasets. Since the experiments on each dataset are independent, we believe that it does not fully explain that our method is universally applicable. Therefore, we combine the three datasets together to form a larger dataset and experiment on it to mitigate this issue.

Construct Validity. Although the performance has been improved, the data which is mislabeled in our datasets is an issue. It is possible that the bug reports used to create the detection model are incorrectly labeled, which may cause errors in our model. In order to alleviate this problem, we make experiments with three datasets from three different projects, which are labeled by different groups of people.

8 CONCLUSION AND FUTURE WORK

In this paper, we propose a novel approach using DC-CNN for duplicate bug report detection. We concatenate two bug reports represented by two single-channel matrices into a bug report pair represented by a dual-channel matrix. Then, the dual-channel matrix is fed into the CNN model to extract their hidden semantics and features. We have investigated the performance of DC-CNN on three public datasets including Open Office, Eclipse and Net Beans and a larger dataset combined by these three projects. We evaluate DC-CNN's performance against the state-of-the-art deep-learning-based approaches[8, 12] for duplicate bug report detection. In summary, our conclusion is as follows:

(1) Compared with the state-of-the-art deep-learning-based approaches, DC-CNN performs better for duplicate bug reports detection.
(2) Compared with SC-CNN in terms of $Accuracy$, $Recall$, $Precision$ and $F1-Score$, DC-CNN is better. This indicates that our proposed dual-channel bug report pair representation is effective.
(3) We make a comparative study by removing structured information compared with our default approach. We find that it is more effective by feeding both structured information and unstructured information to CNN.
(4) We make a cross-project setting experiment. The experimental results indicate that when a project lacks labeled data, we can use the existing labeled data from other appropriate project(s) to train the detection model.

In the future, we will investigate how to make use of more structured information to improve our approach. Additionally, more empirical studies can be performed to validate our approach on both open source and industrial projects.

ACKNOWLEDGMENTS

The work described in this paper was partially supported by the National Key Research and Development Project (Grant no. 2018YFB2101201), the National Natural Science Foundation of China (Grant no. 61602504), the Fundamental Research Funds for the Central Universities (Grant no. 2019CDYGYB014) and the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE200100021).

REFERENCES

[1] Karan Aggarwal, Finbarr Timbers, Tanner Rutgers, Abram Hindle, Eleni Stroulia, and Russell Greiner. 2017. Detecting duplicate bug reports with software engineering domain knowledge. *Journal of Software: Evolution and Process* 29, 3 (2017), e1821.

[2] Anahita Alipour, Abram Hindle, and Eleni Stroulia. 2013. A contextual approach towards more accurate duplicate bug report detection. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 183–192.

[3] John Anvik, Lyndon Hiew, and Gail C Murphy. 2005. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. ACM, 35–39.

[4] Prasad V Bagal, Sameer Arun Joshi, Hanlin Daniel Chien, Ricardo Rey Diez, David Cavazos Woo, Emily Ronshien Su, and Sha Chang. 2019. Duplicate bug report detection using machine learning algorithms and automated feedback incorporation. US Patent App. 16/383,405.

[5] Andrzej Białecki, Robert Muir, Grant Ingersoll, and Lucid Imagination. 2012. Apache lucene 4. In *SIGIR 2012 workshop on open source information retrieval*. 17.

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

[7] Satya Prateek Bommaraju, Anjaneyulu Pasala, and Shivani Rao. 2018. System and method for detection of duplicate bug reports. US Patent 9,990,268.

[8] Amar Budhiraja, Kartik Dutta, Raghu Reddy, and Manish Shrivastava. 2018. 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*. 193–194.

[9] Amar Budhiraja, Kartik Dutta, Manish Shrivastava, and Raghu Reddy. 2018. Towards Word Embeddings for Improved Duplicate Bug Report Retrieval in Software Repositories. In *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*. ACM, 167–170.

[10] Yguarata Cerqueira Cavalcanti, Eduardo Santana de Almeida, Carlos Eduardo Albuquerque da Cunha, Daniel Lucredio, and Silvio Romero de Lemos Meira. 2010. An initial study on the bug report duplication problem. In *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE, 264–267.

[11] Catarina Costa, Jair Figueiredo, Leonardo Murta, and Anita Sarma. 2016. TIPMerge: recommending experts for integrating changes across branches. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 523–534.

[12] Jayati Deshmukh, Sanjay Podder, Shubhashis Sengupta, Neville Dubash, et al. 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 115–124.

[13] Yuanrui Fan, Xia Xin, Lo David, and Hassan Ahmed E. [n. d.]. Chaff from the Wheat: Characterizing and Determining Valid Bug Reports. *IEEE Transactions on Software Engineering* ([n. d.]), 1–1.

[14] Ying Fu, Meng Yan, Xiaohong Zhang, Ling Xu, Dan Yang, and Jeffrey D Kymer. 2015. Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. *Information and Software Technology* 57 (2015), 369–377.

[15] Lyndon Hiew. 2006. *Assisted detection of duplicate bug reports*. Ph.D. Dissertation. University of British Columbia.

[16] Pieter Hooimeijer and Westley Weimer. 2007. Modeling bug report quality. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 34–43.

[17] Nicholas Jalbert and Westley Weimer. 2008. Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 52–61.

[18] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Generating duplicate bug datasets. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 392–395.

[19] Hoa T Le, Christophe Cerisara, and Alexandre Denis. 2018. Do convolutional networks need to be deep for text classification?. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

[20] Dean Lee, Vincent Siu, Rick Cruz, and Charles Yetman. 2016. Convolutional neural net and bearing fault analysis. In *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 194.

[21] Johannes Lerch and Mira Mezini. 2013. Finding duplicates of your yet unwritten bug report. In *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 69–78.

[22] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. 2015. Support vector machines and word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE, 136–140.

[23] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101* (2016).

[24] Tao Liu, Zheng Chen, Benyu Zhang, Wei-ying Ma, and Gongyi Wu. 2004. Improving text classification using local latent semantic indexing. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE, 162–169.

[25] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[26] Marc Moreno Lopez and Jugal Kalita. 2017. Deep Learning applied to NLP. *arXiv preprint arXiv:1703.03091* (2017).

[27] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. 2012. 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, 70–79.

[28] Xin Rong. 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738* (2014).

[29] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 499–510.

[30] Nicolas Serrano and Ismael Ciordia. 2005. Bugzilla, ITracker, and other bug trackers. *IEEE software* 22, 2 (2005), 11–13.

[31] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 253–262.

[32] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 45–54.

[33] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

[34] Ashish Sureka and Pankaj Jalote. 2010. Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific Software Engineering Conference*. IEEE, 366–374.

[35] D Swapna and K Thammi Reddy. 2016. Duplicate Bug Report Detection of User Interface Bugs using Decision Tree Induction and Inverted Index Structure. (2016).

[36] Yuan Tian, Chengnian Sun, and David Lo. 2012. Improved duplicate bug report identification. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 385–390.

[37] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*. ACM, 461–470.

[38] Meng Yan, Ying Fu, Xiaohong Zhang, Dan Yang, Ling Xu, and Jeffrey D Kymer. 2016. Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project. *Journal of Systems and Software* 113 (2016), 296–308.

[39] Meng Yan, Xiaohong Zhang, Dan Yang, Ling Xu, and Jeffrey D Kymer. 2016. A component recommender for bug reports using discriminative probability latent semantic analysis. *Information and Software Technology* 73 (2016), 37–51.

[40] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. 2008. Text classification based on multi-word with support vector machine. *Knowledge-Based Systems* 21, 8 (2008), 879–886.

[41] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*. 649–657.

[42] Jian Zhou and Hongyu Zhang. 2012. Learning to rank duplicate bug reports. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 852–861.

[43] Jie Zou, Ling Xu, Mengning Yang, Meng Yan, Dan Yang, and Xiaohong Zhang. 2016. Duplication Detection for Software Bug Reports based on Topic Model. In *2016 9th International Conference on Service Science (ICSS)*. IEEE, 60–65.

[44] Jie Zou, Ling Xu, Mengning Yang, Xiaohong Zhang, Jun Zeng, and Sachio Hirokawa. 2016. Automated duplicate bug report detection using multi-factor analysis. *IEICE TRANSACTIONS on Information and Systems* 99, 7 (2016), 1762–1775.