2017 IEEE International Conference on Software Maintenance and Evolution

Towards Accurate Duplicate Bug Retrieval using Deep Learning Techniques

Jayati Deshmukh, Annervaz K M, Sanjay Podder, Shubhashis Sengupta, Neville Dubash
Accenture Technology Labs
{jayati.deshmukh, annervaz.k.m, sanjay.podder, shubhashis.sengupta, neville.dubash}@accenture.com

***Abstract*—Duplicate Bug Detection is the problem of identifying whether a newly reported bug is a duplicate of an existing bug in the system and retrieving the original or similar bugs from the past. This is required to avoid costly rediscovery and redundant work. In typical software projects, the number of duplicate bugs reported may run into the order of thousands, making it expensive in terms of cost and time for manual intervention. This makes the problem of duplicate or similar bug detection an important one in Software Engineering domain. However, an automated solution for the same is not quite accurate yet in practice, in spite of many reported approaches using various machine learning techniques. In this work, we propose a retrieval and classification model using Siamese Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM) for accurate detection and retrieval of duplicate and similar bugs. We report an accuracy close to 90% and recall rate close to 80%, which makes possible the practical use of such a system. We describe our model in detail along with related discussions from the Deep Learning domain. By presenting the detailed experimental results, we illustrate the effectiveness of the model in practical systems, including for repositories for which supervised training data is not available.**

***Index Terms*—Information Retrieval, Duplicate Bug Detection, Deep Learning, Natural Language Processing, Word Embeddings, Siamese Networks, Convolutional Neural Networks, Long Short Term Memory**

I. Introduction and Motivation

Software Maintenance is a major part of software development life cycle. Maintenance activities account for over two-thirds of the life cycle cost of a software system [Boehm and Basili, 2005]. All large software projects host a bug tracking system like Bugzilla[1] to manage and keep track of bugs and their fixes. The testers and the end users of the software system report bugs in such systems, in a highly non-coordinated fashion, leading to the reporting of duplicate bugs. Moreover, a bug that had occurred previously and was fixed, could later resurface for the same or different reason. In both the scenarios, the effective mapping of a bug to a previously reported similar bug is essential for efficiency reasons. Such a mapping will help to avoid unnecessary rework by engineers fixing the bug and to carry out appropriate *Bug Prioritization and Triage*. The identification of similar bugs (may not be exact duplicates) may also help the support engineers to identify corresponding fixes from past analysis, thus increasing their productivity. Manual identification of duplicate or similar bugs is time consuming, and hence costly. It also requires considerable expertise in the project domain to carry out the task. In large projects like ThunderBird[2] the percentage of duplicate bugs account for a total of almost 50% of the total bugs reported [Cavalcanti et al., 2010], accounting for considerable wastage of time and resources. In these contexts, automation of duplicate bug identification and retrieval of similar past bugs becomes important in software maintenance.

Typical bug tracking systems maintain a master bug list. When a new bug is entered in the system, the support engineers analyze the bug and conduct searches in the repository for potential duplicates of the bug bearing similar signatures. If no duplicate is found; the bug is added to the master list; else it is marked as a duplicate. A typical bug report contains structured information like product / module, component, version, date on which the bug is reported etc. along with unstructured natural language descriptions of the bug. There are typically two kinds of descriptions - short and long. The inherent variability of language to express the same bug is what makes the problem of duplicate bug detection relatively hard. Many approaches have been proposed in the literature for automating the identification of duplicate bug and retrieving similar bugs using machine learning and information retrieval techniques [Alipour et al., 2013; Sun et al., 2010, 2011; Wang et al., 2008]. In spite of the considerable volume of work, the reported accuracy numbers are relatively low [Sun et al., 2011] [Kaushik and Tahvildari, 2012]. It is also not clear empirically if these techniques have been successfully adopted in practice.

Unsupervised feature learning and Deep Learning [Bengio, 2009; LeCun et al., 2015] based on Neural Networks have gained prominence in the last few years. State-of-the-art *Neural Network* models and appropriate algorithms to train these models have been proposed for multitude of tasks in Computer Vision, Natural Language Processing, Speech Recognition etc. In the area of natural language processing; the recent deep learning models have been proven superior to classical machine learning approaches in many tasks like Part of Speech Tagging, Question Answering, Sentiment Analysis, Document Classification [Kumar et al., 2015] etc.

In this work, we seek to leverage the advances in deep learning for natural language processing to improve the accuracy of duplicate / similar bug detection and retrieval and propose a practically usable system. We use Siamese variants of

[1] www.bugzilla.org

[2] www.mozilla.org/en-US/thunderbird/

978-1-5386-0992-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSME.2017.69

Convolutional Neural Networks and *Recurrent Neural Network* to propose a model for duplicate / similarity detection from bug signatures (structured and unstructured information). To the best of our knowledge, this is first attempt of using Deep Learning techniques for duplicate bug detection.

The main contributions of this paper can be summarized as follows,

1) A deep learning model is proposed based on state-of-the-art constructs for effective detection and retrieval of duplicate / similar bugs. The model can make use of the structured as well as unstructured information available in the bug signature.
2) The proposed model uses Siamese Neural Networks, trained on max margin objective to distinguish similar from non-similar bugs. The complete model objective combination is novel.
3) We present detailed experimental results on 3 large published datasets [Lazar et al., 2014]. The models yield accuracy close to 90%, and recall rate close to 80%, thereby outperforming the previous reported approaches and enabling the use of such a system in practice.
4) We illustrate the models can be used effectively on software projects where training data is not directly available for supervised training, by training it on samples from other projects of the same domain.

Rest of the paper is organized as follows. We cover related work in the next section II . Our model along with the minimal deep learning prerequisites are covered in detail in section III . Various experimental results and detailed analysis is presented in section IV . Finally, we conclude in section V by providing some future directions.

II. Previous Work

Detection and retrieval of duplicate and similar bugs is a well studied problem in the literature with many proposed approaches. At the onset, we can classify the approaches as supervised classification and unsupervised information retrieval approaches.

Unsupervised Approaches: Typical strategy used here is to create a numerical similarity scoring function like cosine similarity [Deza and Deza, 2009] between appropriate vector representation (like one hot encoding) of bugs; and rank the bugs based on similarity score - using which top k similar bugs can be retrieved. A variety of similarity scores are published in the literature. Variants of advanced scoring functions like BM25F [Robertson et al., 2004] famous in information retrieval community is used in many places [Sun et al., 2011]. In [Sureka and Jalote, 2010] hand-crafted features like character N-grams are used for computing the similarity score. Yet other unsupervised approaches use topic modeling techniques like Latent Dirichlet Allocation (LDA) [Nguyen et al., 2012] for calculating the similarity score. [Kaushik and Tahvildari, 2012] is a comparative study (corresponding masters thesis [Kaushik, 2012]) of main IR approaches for duplicate bug detection and it reports recall rate (ratio of number of duplicates detected to the total number of bugs which have "true" duplicates in the test set) close to 60% for most of the unsupervised approaches.

Supervised Approaches: Supervised approaches primarily pose the problem as a binary classification problem. Given two bugs, the classifier has to predict whether they are similar or not. Supervised training samples are created for training a binary classifier for this purpose. Works in these lines [Sun et al., 2010] use traditional classifiers like Support Vector Machines on hand-selected features for training. Approaches where supervised training is done for computing the similarity score effectively without posing it as a classification problem are also proposed in literature [Sun et al., 2011] . Some approaches like [Alipour et al., 2013] [Wang et al., 2008] make use of contextual information like execution and stack traces etc along with the bug reports for improving the recall rate. Recall rates for most supervised approaches are also close to 60% [Sun et al., 2010] only.

III. Deep Learning Approach

In this section, we describe the proposed deep learning model for duplicate / similar bug detection and retrieval in detail. We use *Word Embeddings* to convert natural language bug descriptions into a numerical representation. These numerical representations are then encoded using a *Long Short Term Memory* and a *Convolutional Neural Network* model. Before explaining our approach, let us discuss briefly these three basic deep learning constructs as a prerequisite.

A. Deep Learning Prerequisites

1) Word Embeddings: The first challenge encountered in applying machine learning models for natural language processing is to find a correct numerical representation for words. *You shall know a word by the company it keeps (Firth, J. R. 1957:11)*, is one of the most influential ideas in natural language processing. Multiple models for representing a word as a numerical vector, based on the context it appears, stem from this idea. Such vector representations for words have been utilized in multiple ways, including the well known Latent Semantic Analysis (LSA) [Dumais, 2004]. Vector representations for words in the context of neural networks was proposed by [Bengio et al., 2003]. In this work, each word in the vocabulary is assigned a *Distributed Word Feature Vector* $\in \mathcal{R}^m$. The probability distribution of word sequences, $P(w_t|w_{t-(n-1)}, .., w_{t-1})$, is then expressed in terms of these word feature vectors. The word feature vectors and parameters of the probability function (which is a neural network) are learned together by training a suitable feed-forward neural network [Haykin, 1998] to maximize the log-likelihood of the text corpora. Each occurrence of a text snippet of fixed window size (defined hyper parameter) is considered here as a training sample. Inspired from this model [Mikolov et al., 2013a] proposed two new models - continuous bag of words (CBOW) and skip-gram model, popularly known as *Word2Vec* models. The CBOW architecture tries to predict the current word given the previous and next surrounding words, discarding the word order, in a fixed context window.

Skip-gram model predicts the surrounding words given the current word. These models have better training complexity, and thus can be used for training on large corpus. The vectors generated by these models on large corpus have shown to capture subtle semantic relationships between words, by simple vector operations on them [Mikolov et al., 2013b]. For example, *Vector('King')-Vector('Man')+Vector('Woman')* is closest to *Vector('Queen')*. Similarly *Vector('Germany')-Vector('Berlin')* is approximately equal to *Vector('France')-Vector('Paris')*. Another variant named *GloVe* word vectors was proposed by [Pennington et al., 2014]. This model tries to create word vectors in such a way that if we take dot product of two vectors, it will closely resemble the co-occurrence count of the corresponding words. The model is more effective compared to *Word2Vec* models for capturing semantic regularities on smaller corpus. We have tried both embeddings in the experiments.

2) Recurrent Neural Network and Long Short Term Memory (LSTM): The basic idea behind Recurrent Neural Networks (RNN) is to make use of the information present in a given sequence like text, where text is just a sequence of words. Given a sequence of words, a numerical representation (GloVe or Word2Vec vectors) of the word is fed to a neural network and the output is computed. While computing the output for the next word, the output from the previous word (or time step) is also considered. RNNs are called recurrent because they perform the same computation for every element of a sequence using the output from previous computations. At any step RNN performs the following computation,

$$\mathrm{RNN}(t_i) = f(W * x_{t_i} + U * \mathrm{RNN}(t_{i-1})),$$

where W and U are the parameters of the model, and f is any nonlinear function. The bias terms are left out here and have to be added appropriately. $RNN(t_i)$ is the output at i^{th} timestep, which can either be utilized as is, or can be fed again to a parameterized construct such as softmax [Bishop, 2006], depending on the task at hand. The training is done by formulating a loss objective function based on the outputs at all or some of the timesteps, and trying to minimize the loss. The vanilla RNNs explained above have difficulty in learning long term dependencies in the sequence via gradient descent training [Bengio et al., 1994]. Also training vanilla RNNs is shown to be difficult because of vanishing and exploding gradient problems [Pascanu et al., 2013]. Long short term memory (LSTM) [Hochreiter and Schmidhuber, 1997], a variant of RNN is shown to be effective in capturing dependencies and easier to train compared to vanilla RNNs. Multiple kinds of LSTMs have been proposed in the literature. Please refer [Greff et al., 2015] for a comprehensive survey of LSTM variants and motivations behind. The version we are using is explained next.

A LSTM module has three parameterized gates, input gate(i), forget gate(f) and output gate(o). A gate g operates by

$$g_{t_i} = \sigma(W^g * x_{t_i} + U^g * h_{t_{i-1}}),$$

where W^g and U^g are the parameters of the gate g, h_{t-1} is the hidden state at the previous time step and σ stands for the sigmoid function. All the three gates have the same equation form and inputs, but they have different set of parameters. Along with hidden state, LSTM module also has a cell state. The updation of the hidden state and cell state at any timestep is controlled by various gates as follows,

$$C_{t_i} = f_{t_i} * C_{t_{i-1}} + i_{t_i} * \tanh(W^C * x_{t_i} + U^C * h_{t_{i-1}})$$

and

$$h_{t_i} = o_{t_i} * \tanh(C_{t_i}),$$

where W^C and U^C are again parameters of the model. The key component of the LSTM is the cell state. LSTM has the capability to modify and retain the content on the cell state as required by the task, using the gates and hidden states. A forward LSTM is the one taking the input sequence as it is. A backward LSTM is the one taking the input in the reverse order. A backward LSTM is used to capture the dependencies of a word on future words in the original sequence. A concatenation of a forward LSTM and a backward LSTM is known as bi-directional LSTM (bi-LSTM), which we are using in our model.

3) Convolutional Neural Networks: LSTMs captures dependencies in a sequence, typically over a small range. This is good enough to capture the information in the short description of the bug. However the bug description can be very long in the long description, comparable to a document. We use Convolutional Neural Network(CNN) for handling such long text. CNN although first proposed in the context of images [Krizhevsky et al., 2012], is shown to be effective for many text classification tasks recently [dos Santos and Gatti, 2014; Kim, 2014].

A convolutional filter takes a part of an input, computes a function, outputs the computed value; and then takes another part of the input, computes the same function and outputs it. The filter goes on repeating the same process till the complete input is covered. All the outputs are then concatenated to form a smaller representation of the input or a pooling layer is applied. A max pooling layer aggregates the outputs by taking the max, and mean pooling layer takes the mean of the outputs. In the context of an image, convolutions are in 2 dimensions because image is typically 2D. In the context of text we have one dimensional input which is the concatenation of the numerical representation of the words of the text. Convolutional filter starts from the beginning of the text, considers a certain length of the text (called Window Size or Filter Size), does a function computation with the input and outputs the value as given in the equation below. It then moves to the right by a certain amount (called Stride length) and repeats the same computation. The process goes on till the right end of the input (or padded input) is encountered. The outputs are then concatenated or a pooling layer is applied. The same operator can be applied again(with same or different set of parameters), considering the reduced output

as input. Such stacked convolutional architectures are called Deep Convolutional Networks. The final encoding generated by such architectures can be used for the final prediction task.

$$For\ all\ i \in \{0, s, 2s, ..\},\ C_i = f(W * x_{i:i+h-1} + b)$$

The above equation depicts the application of the filter on the input from i^{th} word considering h words, here h is the window size and s is the stride length. $W \in \mathcal{R}^{hk}$ is the model parameters, k is the dimension of the individual word vectors. f is any non linearity function like sigmoid, b is the bias term. Various C_i's are concatenated together to form the output representation and then pooling layer or another convolutional layer is applied as discussed earlier.

B. The Proposed Model

A typical bug description has 3 main information parts. Structured information (like component information), a short description and a long description. These 3 main parts are processed separately using different neural network architectures, capable of addressing their properties and to learn pertinent features from them. As these parts are processed separately, the model can be used (with small modification and training) in practice even if one component is not available for certain projects.

The structured information(after coverting to an appropriate numerical representation) of a bug b is encoded using a vanilla single layer neural network ($\sigma(W^T \cdot b)$) to output $\mathcal{I}_b$. The long and the short descriptions of the bug are first converted into its numerical representation using trained word vectors. The short description is then encoded by a bi-LSTM (mean pooled across the sequence) to output $\mathcal{S}_b$. The long descriptions are encoded by a CNN, with a convolutional layer and a max pooling layer. Multiple CNNs are applied with varying filter sizes. The output of all of them are concatenated to form the representation $\mathcal{L}_b$. For the bug b, final encoding $\mathcal{F}_b$ is formed by concatenating all of the individual component encodings, $\mathcal{F}_b = \mathcal{I}_b : \mathcal{S}_b : \mathcal{L}_b$.

We have trained two models - one for classification and one for retrieval - using these encodings of the bugs. First let us see the retrieval model. The objective of encoding the bugs is to help us to identify similar bugs. Thus encodings should be learned in such a way that it provides high similarity for similar bugs and very low similarity for dissimilar bugs. This notion is formulated as a max margin training loss objective as given below:

$$Max\{0, M - Cosine(\mathcal{F}_b, \mathcal{F}_{b+}) + Cosine(\mathcal{F}_b, \mathcal{F}_{b-})\}$$

where M is a hyperparameter typically set to 1, b is any bug, b+ is its duplicate and b- is any random non-duplicate bug of b from the data.

The model is trained for minimizing this loss averaged across the training samples, using back propagation algorithm and stochastic gradient descent [Bottou, 2012; Haykin, 1998]. The model, trained using this objective is in effect trying to place the bug in a latent feature space in such a way that duplicate bugs will have high similarity score (illustrated by t-SNE plot of the encodings of the bugs in section IV-F) and non-duplicate bugs will have a very low similarity score. The discriminative features which facilitate it are learned for the encodings automatically from the supervised training using the max margin loss.

A schematic diagram of the full model is given in Figure 1 . Here each training sample has three bugs, two taken at a time $(b, b+)$ and $(b, b-)$ and the same network is encoding all bugs. This can be viewed as two networks, one encoding one bug and another encoding the other, but with the weights tied down. These kind of architectures are typically known as Siamese Networks [Mueller and Thyagarajan, 2016] in deep learning literature. So our model can be viewed as combination of Siamese LSTM and CNN. The dotted lines in Figure 1 depict shared weights among the models.

When a new bug n is encountered, its encoding $\mathcal{F}_n$ is first computed from the trained model and then the similarity with each of the bug in the master set is computed, based on cosine similarity. Duplicate bugs can then be reported, either based on top k similar bugs or by putting a threshold on the similarity value. Recall rate and accuracy values will vary if we change the k or the threshold of similarity value. Detailed results are reported in the next section.

The classification model is very close to the retrieval model for the encoding part. Main difference is in the objective used for training. Here, we are posing the problem of duplicate bug identification as a binary classification problem, given two bugs the classes are 'duplicate' and 'non-duplicate'. The encodings, $\mathcal{F}_a, \mathcal{F}_b$ of the two bugs a, b are concatenated and passed into a two layer neural network and softmax function is applied on the output of the top layer to do the final binary prediction task. Cross entropy [Bishop, 2006] between the supervised label q and the predicted label p, $H(q, p)$, averaged across the training samples is minimized to learn the various model parameters(mainly of the final fully connected layer, other layers are pretrained by retrieval model discussed earlier) using stochastic gradient descent. Classification model equations are given below.

$$\text{label}_{\text{predict}}(a, b) = \text{softmax}(\sigma\left((W_2^i)^T * \sigma((W_1^i)^T[\mathcal{F}_a : \mathcal{F}_b])\right))$$

$$\text{loss} = H(\text{label}_{\text{supervised}}(a, b), \text{label}_{\text{predict}}(a, b))$$

When a new bug n is encountered, it is paired with all the bugs in the master set and trained classification model is used to predict if the pairs are duplicate or not. Output value of the duplicate predicting neuron after applying softmax function is used to rank the bugs for the similar bug retrieval. Detailed results are reported in the next section.

IV. Experiments & Evaluation

This section describes the evaluation of the classification and retrieval model described in section III-B. All the models were implemented in TensorFlow [Abadi et al., 2015] - an

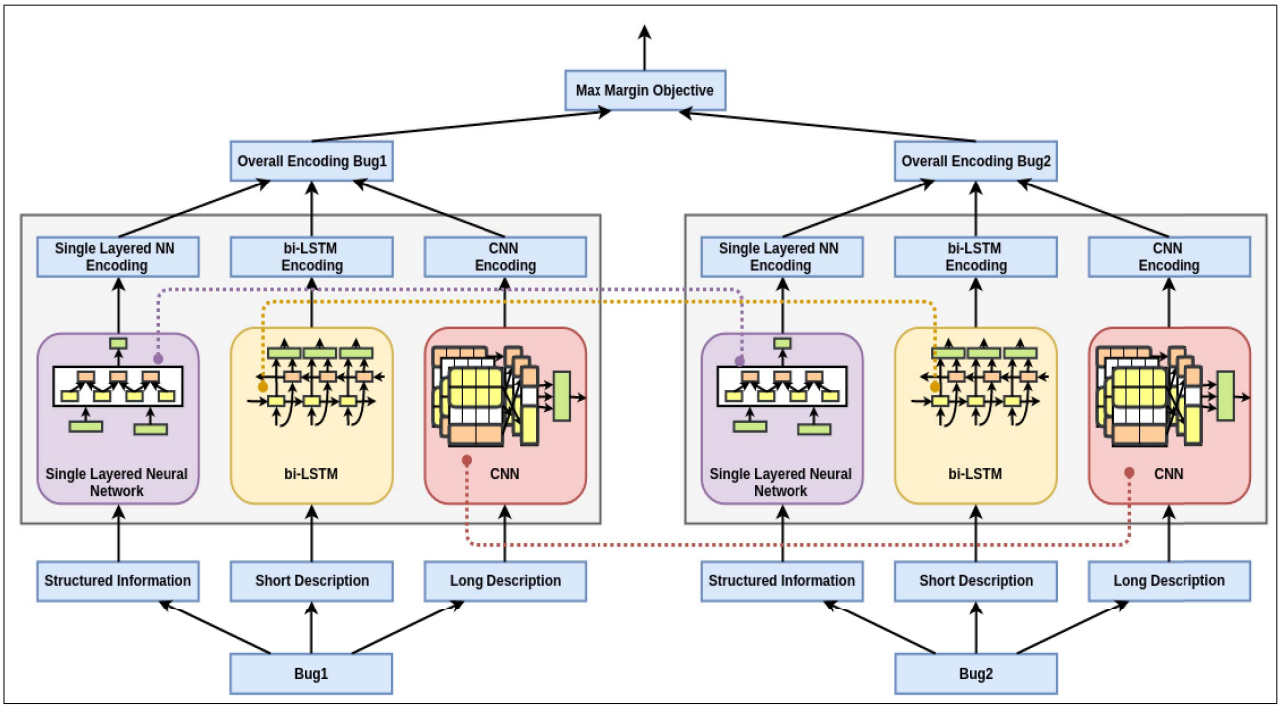

Fig. 1. The Complete Model

TABLE I
MODEL PARAMETERS

Parameter Name	CNN	LSTM
Word Vector Dimension	300	300
Sequence Length	-	100
Filter Sizes	3,4,5	-

TABLE II
SAMPLE BUG

”bug_id”	”2521”
”product”	”Writer”
”description”	”Opening a file from the file history that has been moved or deleted causes an error dialogue to pop up saying it doesn’t exist etc. Upon clicking OK the dialogue comes up again. This continues indefinitely meaning the application has to be terminated.”
”bug_severity”	”trivial”
”dup_id”	”2268”
”short_desc”	”Opening a recent doc that has since been moved / deleted causes unfulfilled loop”
”priority”	”P3”
”version”	”641”
”component”	”ui”
”delta_ts”	”2003-09-08 16:56:16 +0000”
”bug_status”	”CLOSED”
”creation_ts”	”2001-12-12 17:00:00 +0000”
”resolution”	”DUPLICATE”

open-source library for numerical computation for deep learning. All experiments were carried on a Dell Precision Tower 7910 server with Nvidia Quadro M6000 GPU. The models were trained using Adam’s Optimizer [Kingma and Ba, 2014] in a stochastic gradient descent [Bottou, 2012] fashion for 100 epochs. Various model parameters are mentioned in Table I.

A. Datasets and Preprocessing

We used three large bug datasets collected from Open Office, Eclipse and Net Beans projects, which have been curated and published by [Lazar et al., 2014]. Eclipse and Net Beans are open source integrated development environments for Java and Open Office is an open-source productivity software like Microsoft Office. We also created a larger dataset by combining the bugs from all the three datasets and have referred it as ’Combined’ dataset. The combined dataset was formed to experiment with larger training samples.

The datasets have a variety of fields to describe the bug properties, such as, bug_id, product, description, bug_severity, short description, priority, dup_id, version, component, status, resolution etc. A sample bug with all the bug properties is shown in Table II . We have used ’bug_id’ to uniquely identify the bugs, ’description’ which describes the bug in detail, and ’dup_id’ which is the list of duplicate bugs for a given bug. The datasets are pre-processed to create triplets of the form $(b, b+, b-)$, where b is any bug, $b+$ and $b-$ is duplicate and non-duplicate respectively. Non-duplicates are generated randomly from the dataset. The triplets are used directly to train the retrieval model. From the triplet, $(b, b+)$ and $(b, b-)$ are skimmed to create the training samples for the classification model. Table III shows the number of bug pairs in the train and test splits (using $80-20\%$) for all the datasets.

Basic pre-processing was performed on the

TABLE III
TRAIN TEST SPLIT OF DATASETS

	Train	Test
Open Office	88896	22225
Eclipse	88144	22037
Net Beans	118674	29669
Combined	295714	73931

TABLE IV
ACCURACY OF MODELS

	Retrieval	Classification
Open Office	0.9455	0.8275
Eclipse	0.906	0.7268
Net Beans	0.9127	0.7745
Combined	0.9168	0.8219

bug_descriptions. The text was cleaned by removing all non-alpha-numeric characters. Next, the sentences were passed through named entity recognition (NER) module and then the named entities were replaced with their labels like "PERSON", "LOCATION", "DATE" etc. This was followed by converting all the text to lower case and tokenizing them into words based on spaces. A dictionary was created of all unique words in the union of test and train dataset. Each word in the dictionary was then mapped to its corresponding embedding. We experimented with both GloVe vectors trained[3] on Common Crawl dataset as well as Word2Vec vectors trained[4] on Google news dataset. We used Common Crawl trained GloVe word embeddings for the final reported results as it had a greater number of pre-trained words available as compared to Word2Vec. GloVe word vectors for words which appear in the dataset but don't appear in the common crawl were trained separately using the text extracts from the datasets. The word embeddings are not trained along with the final model.

B. Evaluation Metrics

The recall rate for both the models was calculated as the ratio of original duplicate bugs identified to the total number of bugs which had duplicate in the test set. We used recall rate as the evaluation metric as it is the metric used in most of the earlier work, and thus we could compare our models to the previous techniques to some extent. For the retrieval model the duplicate bugs were retrieved by ordering the bugs in the test dataset according to the cosine similarity between the encodings $\mathcal{F}$ of the bugs. For the classification model output value after applying softmax function of the 'duplicate' predicting neuron is used to order the bugs in the dataset to retrieve top k similar bugs. Accuracy for the classification model is calculated in the standard way. For the retrieval model classification is done based on the threshold value of the cosine similarity of the encodings.

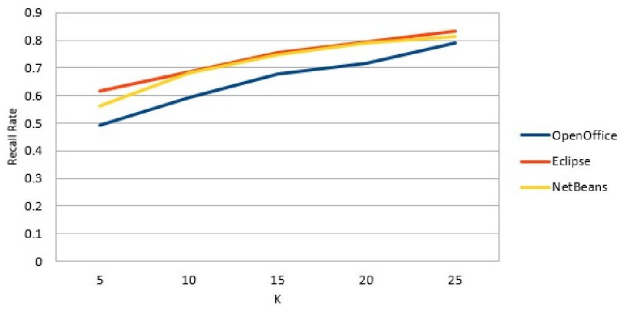

Fig. 2. Recall Rate of Retrieval Model for various values of K

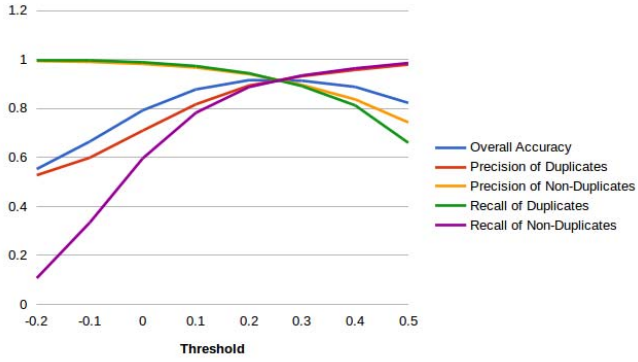

Fig. 3. Classwise Results of Retrieval Model on Combined Dataset

C. Main Results

Table IV shows the accuracy numbers of the retrieval and classification model for all the datasets. The models are able to attain close to 80% accuracy for all datasets and for most datasets close to 90%. Figure 2 depicts how recall rate varies when k is varied for the retrieval model. We are able to achieve a recall rate close to 80% for all the datasets when considering k to be 20 and above which is typical. Figure 3 depicts the class-wise accuracy results i.e. results separately of the duplicate and non-duplicate classes of the combined dataset. The highest accuracy is obtained for a threshold of 0.27.

Many previous works [Sun et al., 2010, 2011; Sureka and Jalote, 2010] report recall rate on partial dated subsets of the Eclipse and Open Office dataset as shown in Table V. It is not clear which subset of these they have used for training and testing, making it difficult for direct comparison. However our accuracy results as described earlier are on complete dataset (without slicing as per dates) and are close to 90% and the recall rate is close to 80% which is better than previously reported numbers for most supervised and unsupervised methods [Kaushik, 2012; Kaushik and Tahvildari, 2012] and discriminative approaches like [Sun et al., 2010]

[3]http://nlp.stanford.edu/data/glove.42B.300d.zip
[4]https://code.google.com/archive/p/word2vec/

TABLE V
PREVIOUS RESULTS

Name	Dataset	Start Date	End Date	Recall Rate (top 20)
Discriminative Model [Sun et al., 2010]	Open Office	2/1/2008	30/12/2008	0.66
	Eclipse	2/1/2008	30/12/2008	0.62
$BM25F_{ext}$ [Sun et al., 2011]	Open Office	1/1/2008	21/12/2010	0.69
	Eclipse	1/1/2001	14/12/2007	0.67
DBTM (IR+TM) [Nguyen et al., 2012]	Open Office	1/1/2008	12/21/2010	0.81
	Eclipse	1/1/2008	12/31/2008	0.87

TABLE VI
INPUT BUG

”product”	”Writer”
”description”	”if you have a table over page long it does not automatically continue onto the next page it does not appear to matter if the table has more than row column also if the table cell is over page it moves the start if the table to the top of the next page i have a cell table for and a pages of information to go into the cell i loose the last pages into the unknown”
”bug_severity”	”trivial”
”short_desc”	”tables going off the first page”
”priority”	”P3”
”version”	”OOO 1.1”
”component”	”ui”
”bug_status”	”CLOSED”

with handcrafted features for the problem. For brevity, in the further discussion we are presenting only the accuracy numbers, instead of both. We noticed the accuracy numbers and recall rates are in proportion, hence all implications made further using accuracy numbers hold true with recall rates as well.

An example of retrieval model result is shown in Table VI and VII . The top 5 similar bugs of the bug shown in Table VI are shown in Table VII. It can be observed that the bug descriptions retreived are fairly similar to that of the original bug where it describes a table not appearing properly when it spans more than one page in Open Office dataset.

D. Cross Training Results

In reality, we do not have supervised training data in majority of industrial software projects. This problem holds us back from using supervised training based models in practice in many software projects. To address this issue, we look at the adaptability of the model trained using supervised training samples from some other project. Table VIII shows the results for this experiment. The model and training data is marked on the left hand side and accuracy for the test dataset marked above is reported. The numbers indicate that it is not necessary that we collect supervised training samples for individual projects, for our model to be used in that project. Instead a model trained on samples from different projects *of the same domain* can be used for highly accurate prediction. This is clear from the good performance of Eclipse dataset trained model on Net Beans and viceversa. Both the projects are from the same genre and domain, *Integrated Development Environment*. However this is not the case for Open Office dataset as it is from a different domain. This illustrates that generic semantic features of the domain are learned by the model, which makes this cross project transfer within the same domain possible.

E. Useful Information Spread

Many previous works [Sun et al., 2011][Alipour et al., 2013] use heuristics based on the structured information available in bug data, like component id to prune the search space and to increase the accuracy numbers. But these information many not be reported for many bugs in practical cases. This information can vary across the projects as well. Such heuristics will fail in these cases. In our model we have made provisions to make use, if this structured information is available. But our model learns rich information from the text data directly. The accuracy numbers of the retrieval model when only the short and long description is used, excluding the structured information is given in the Table IX. The accuracy numbers with (in Table IV) and without using structured information are very close, illustrating that model can be used even when the structured information is not available, typical in practical scenarios. It also illustrates that the model learns semantically rich features from the text directly to identify duplicate / similar bugs.

F. Encodings of the Bugs

Conceptually the retrieval model is trying to learn encodings of the bug in such a way that duplicate bugs in the latent feature space are placed in the same direction. Figure 4 shows the 2 dimensional t-SNE [Maaten and Hinton, 2008] plot of encodings ($\mathcal{F}$) of some random duplicate bugs. Original bugs are red coloured, duplicate bugs are green coloured and non-duplicate bugs are blue coloured. Original bugs are labelled with the original bug id, *O_bug*. Duplicate bugs are labelled as *O_bug+[id]* and non-duplicate bugs are labelled as *O_bug-[id]* , where the ids are the actual identifiers from the Open Office dataset. As one can notice, the angle between the original bug and duplicate bug is low and the angle between original bug and non-duplicate bug is comparatively high. A sample long description of identified duplicate bug pair by the model is given in Figure 5. One can see that the actual words used are very different, however the model learns rich semantic features out of the text to identify that they are duplicate.

V. CONCLUSION AND FUTURE WORK DIRECTIONS

This is a very basic attempt to leverage the state of the art deep learning models for accurate prediction of duplicate bugs

TABLE VII
TOP 5 DUPLICATES OF INPUT BUG VI

BugId	Description
33540	this bug is produced when the tables are too big they pass under the next page and not on the next page create an ooo writer document for example create a table with columns and rows in the first column and second row type enough page for filling all the page and some more enjoy the beautiful text under the next page this bug was discovered by my mother who i convert to linux few months ago and she has some problem from importing word documents this is not a problem of importing but a problem of the table engine thank you
51112	attaching ms word document that contains a long table many rows when opened in openoffice the long table which starts on page gets truncated as it tries to flow onto page ms word opens the same document and has pages to it openoffice opens it with only pages with the second page being blank i think this bug may relate to but there is also mention in that the problem would be fixed in openoffice by implementing page breaks the problem does not seem to be fixed
43963	hi i am using oo version in win prof i have created a document through writer inserted a table having row and column and went on inserting at the end of page instead of going to next page it contiunes on the first page and the text is going hidden how do i rectify this i am having page ms word document if i open the same document in oo of above version the no of pages it is showing after th page all the pages are blank also the alignment is getting distorted pl guide above issues regards
24904	hi whenever i create a table with a cell which spreads over two pages the text is invisible on the second page the text was corectly and completly inserted into the cell on the first page regards stefan
28170	tables with joined cells in a column are not correctly managed when they use more than one page if the last line has a joined cell it do not continue in the next page it continue over the page limit

TABLE VIII
CROSS TRAINED RESULTS: LEFT HAND SIDE SHOWN TRAIN DATASET AND TOP SHOWS TEST DATASET

		Open Office	Eclipse	Net Beans	Combined
Retrieval	Open Office		0.7153	0.6755	0.7656
	Eclipse	0.7672		0.8051	0.822
	Net Beans	0.7351	0.7917		0.8225
	Combined	0.9439	0.896	0.915	
Classification	Open Office		0.547	0.6238	0.7109
	Eclipse	0.6175		0.7003	0.7063
	Net Beans	0.5715	0.7506		0.6571
	Combined	0.84	0.7048	0.8324	

TABLE IX
ACCURACY OF RETRIEVAL MODEL WHEN USING ONLY PARTS OF THE INFORMATION

	Short Description (bi-LSTM)	Long Description (CNN)
Open Office	0.9373	0.9054
Eclipse	0.8770	0.8652
Net Beans	0.8971	0.8571
Combined	0.8985	0.8504

and retrieval of similar bugs. There are many possible work streams that can be embarked on. We are highlighting some of them below.

1) More Training, Data and Experiments: Only basic training is attempted. Elaborate experimentation is required to train the model effectively, using techniques like batch normalization [Ioffe and Szegedy, 2015] and various other tweaks. Hyperparameter tuning is done only to a certain extent, more can be done in this regard to improve the results further. The training data is also less for a model of this scale, more training data has to be consolidated for various domains (across projects) to train the model.
2) Incorporating attention mechanism: The currently the model does not have any attention mechanism [Xu et al., 2015] [Tan et al., 2015]. Incorporating an appropriate attention mechanism to automatically learn the part of the bug that has to be attended and prioritized, while processing a certain part of duplicate and non duplicate bug, will definitely yield better results.
3) Other models: Attempting models like Tree-LSTMs [Tai et al., 2015] for the encodings of single sentence bug descriptions compared to bi-LSTMs. Paragraph vectors and document vectors [5] [Le and Mikolov, 2014] for the original duplicate/similar bug matching problem should be explored.

We presented a bi-LSTM and CNN based deep learning model for highly accurate duplicate/similar bug detection/retrieval problem. We have presented the results from detailed experiments on publicly available datasets. The accuracy numbers are close to the range of 90% and recall rate is close to 80%, which is the best reported results so far across all the datasets. We have not used any hand crafted features which is the case in most other published works in literature for duplicate or similar bug retrieval problem. We have also illustrated how the model can be used for a project where the supervised training samples are not available, by cross training on projects from the same domain. We have also illustrated

[5]https://radimrehurek.com/gensim/models/doc2vec.html

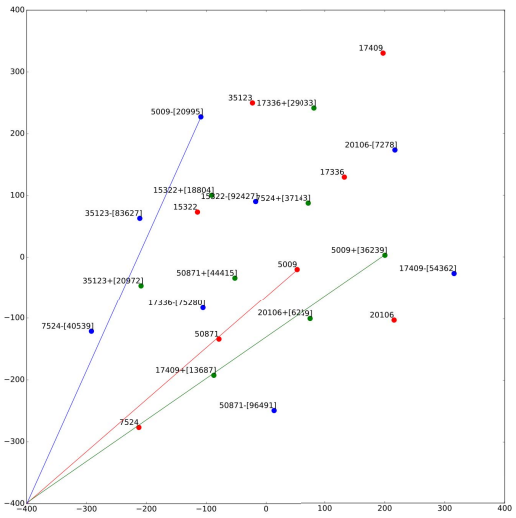

Fig. 4. t-SNE Plot of the encodings ($\mathcal{F}$) of original (Red), duplicate (Green) and non-duplicate (Blue) bugs from Open Office dataset

that the model works very well in the absence of any structured information, which is typically the case in practical settings. To the best of our knowledge this is the first attempt using deep learning techniques for duplicate bug detection problem and the results advocates further work in this direction.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, and Craig Citro et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Anahita Alipour, Abram Hindle, and Eleni Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 183–192. IEEE Press, 2013.

Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Barry Boehm and Victor R Basili. Software defect reduction top 10 list. *Foundations of empirical software engineering: the legacy of Victor R. Basili*, 426:37, 2005.

Lon Bottou. *Stochastic Gradient Tricks*, volume 7700, page 430445. Springer, January 2012. URL `https://www.microsoft.com/en-us/research/publica`

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

Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of Distances*, pages 1–583. Springer, 2009.

Cícero Nogueira dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78, 2014.

Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998. ISBN 0132733501.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Nilam Kaushik. Performance of ir models on duplicate bug report detection: A comparative study. 2012.

Nilam Kaushik and Ladan Tahvildari. A comparative study of the performance of ir models on duplicate bug detection. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 159–168. IEEE, 2012.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*, 2015.

Alina Lazar, Sarah Ritchey, and Bonita Sharif. Generating duplicate bug datasets. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 392–395, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2863-0. doi: 10.1145/2597073.2597128. URL `http://doi.acm.org/10.1145/2597073.2597128`.

Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages

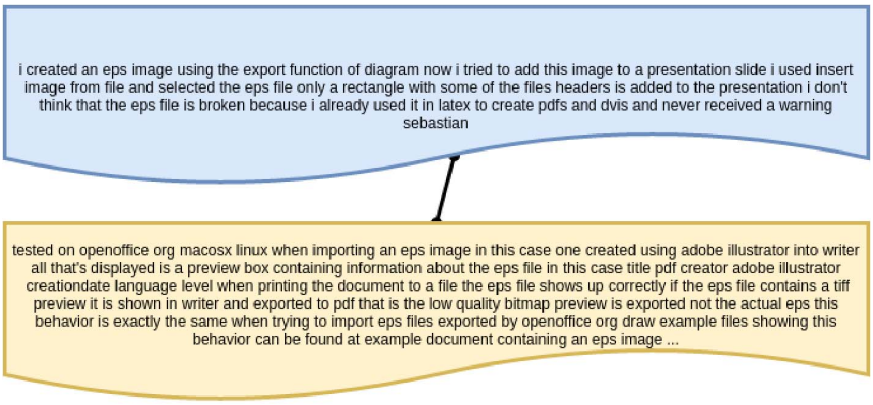

Fig. 5. Long Description of a sample duplicate bug pair

1188–1196, 2014.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9 (Nov):2579–2605, 2008.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, volume 13, pages 746–751, 2013b.

Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, pages 70–79. IEEE, 2012.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49. ACM, 2004.

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

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

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

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Ming Tan, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108*, 2015.

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

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2(3):5, 2015.