POSTER: DWEN: Deep Word Embedding Network for Duplicate Bug Report Detection in Software Repositories

Amar Budhiraja
Microsoft R&D, India
ambudhir@microsoft.com

Kartik Dutta
IIIT-Hyderabad, India
kartik.dutta@research.iiit.ac.in

Raghu Reddy
IIIT-Hyderabad, India
raghu.reddy@iiit.ac.in

Manish Shrivastava
IIIT-Hyderabad, India
m.shrivastava@iiit.ac.in

1 INTRODUCTION AND PROPOSED APPROACH

Bug report filing is a major part of software maintenance. Due to extensive number of bugs filed everyday in large software projects and the asynchronous nature of bug report filing ecosystem, duplicate bug reports are filed. Capturing and tagging duplicate bug reports is crucial in order to avoid assignment of the same bug to different developers. Efforts have been made in the past to detect duplicate bug reports by using topic modelling [2], discriminative methods [5], meta-attributes [6], etc. Recently, Yang *et al.* [8] proposed an approach to combine word embeddings, TF-IDF and meta-attributes to compute bug similarity between two bug reports.

Word embeddings aim at projecting words into low dimensions and learning a dense vector for each word [4]. By learning a dense vector for each word, word embeddings are able to capture semantic and syntactic properties. For example, if we have TF-IDF vectors for 'firefox' and 'browser', it will '1' only at one location for each word and hence, the cosine similarity of 'firefox' and 'browser' would be zero whereas the cosine similarity between these two words for a model trained on Firefox Project's documents is 0.78. Using this notion of word embeddings, we proposed an approach to compute similarity between two bug reports for duplicate bug report detection. We consider each bug report as a text document and use it for training a word embedding model. Using the trained word embedding model, we convert bug reports into vectors and further train a deep neural network on top of these bug report vectors to learn the distributions of duplicate and non-duplicate bug reports bug reports. We call the proposed approach as Deep Word Embedding Network (DWEN) since we trained a deep neural network on top of bug reports vectors created from a word embedding model. Deep Word Embedding Network model has three major steps as shown in Figure 1.

(i). **Word Embedding Training:** Textual components of a bug report (summary and description) were combined as a single document and a word embedding model of dimensions 'D' is trained on these documents.

(ii). **Transformation into Document Vectors of size (D+D):** Using the trained word embedding model, each bug report is converted to a vector of dimensions 'D' by averaging the word vectors contained in the reports. For example, if a bug reports (BR) contains two words w_1 and w_2 and their vectors are $v_{w1} = a_1, a_2, ...a_d$ and $v_{w2} = b_1, b_2, ...b_d$, then the vector for the bug report can be stated using the following equation,

$$v_{BR} = \frac{a_1 + b_1}{2}, \frac{a_2 + b_2}{2}, ... \frac{a_d + b_d}{2} \quad (1)$$

For each pair of bug reports, b_i and b_j, their corresponding vectors, x_i and x_j are concatenated together to create a feature vector of size 'D + D = 2D' for training of a classification deep neural network. The label for each feature vector is 0 if the bug reports are non-duplicates and 1 if the bug reports are duplicates.

(iii). **Training of the deep neural network:** Using the joint bug report vectors created in step (ii), a deep neural network is trained by means of stochastic gradient descent. This neural network learns the distributions of duplicate and non-duplicate bug reports. DWEN learns the probability of similarity of the given pair of bug reports i.e. it outputs a number between 0 and 1 indicating the degree of similarity for the input concatenated vectors. The reason it outputs a number between 0 and 1 is because we trained it with labels 0 and 1, where 0 is the label for non-duplicate reports and 1 is the label for duplicate reports.

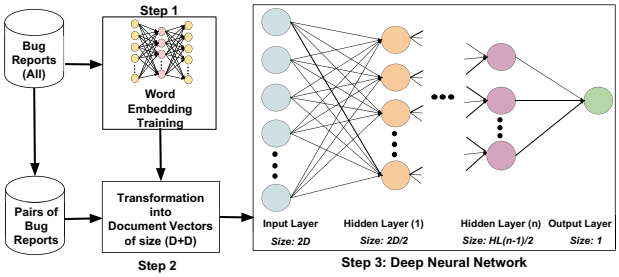


Figure 1: Deep Word Embedding Network Model

Through this three step approach, the proposed model, DWEN, is able to capture syntactic and semantic properties of bug reports by means of the word embddings component and is also able to identify the syntactic and semantic signals which are common amongst two duplicate and non-duplicate bug reports by means of the deep neural network component. In the testing phase, the trained DWEN model is used to compute a similarity between a pair of bug reports. To compute this similarity via DWEN, both bug reports are converted to vectors and concatenated as discussed in point (ii) above and are given as input to DWEN which outputs a similarity score between 0 and 1.

Proposed Architecture: The proposed end-to-end architecture is shown in Figure 2. Firstly, the proposed model DWEN is trained on the bug report database as discussed earlier in this section. For

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195092

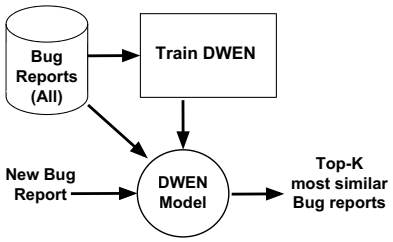


Figure 2: End-to-End Architecture for Duplicate Bug Report Detection based on DWEN: Cylinder box indicates database, circle indicates a model, square box signifies a process and input/output are shown without any boundaries.

Table 1: Results: Recall Rate-1, 5, 20

	Firefox Project			Open Office Project		
Approach	**RR-1**	**RR-5**	**RR-20**	**RR-1**	**RR-5**	**RR-20**
BM25F	0.142	0.256	0.313	0.105	0.239	0.310
LDA [1]	0.067	0.135	0.267	0.128	0.219	0.389
Sureka *et al.* [7]	0.221	0.356	0.490	0.213	0.359	0.493
Yang *et al.* [8]	0.112	0.204	0.301	0.075	0.159	0.277
DWEN	**0.254**	**0.517**	**0.702**	**0.219**	**0.559**	**0.770**

a new bug report, we compute the similarity of the report with all existing reports using the proposed DWEN model. For a pair of bug reports DWEN outputs the degree of similarity (between 0.0 to 1.0). We compute this similarity between the input report and all other bug reports in the corpus and extract the top-K bug reports with the highest similarity to be shown to a triager for identification and tagging of duplicate bug reports.

2 EXPERIMENTS

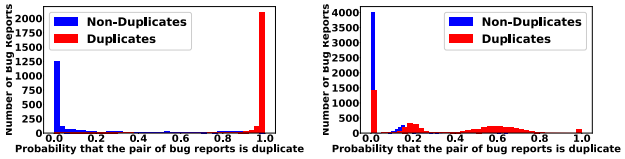
In this section, we detail our experimental setup and results.

Experimental Setup:

We have used the Mozilla Project's and Open Office Project's duplicate bug report datasets generated by Lazar *et al.* [3]. Mozilla Project's dataset has over 700000 reports out of which there are over 140000 duplicate bug report pairs. Open Office Project has over 100000 bug reports with over 19000 tagged duplicate bug report pairs.

We compare BM25F [9] as an information retrieval baseline, LDA [1] as a topic modelling baseline, the approach proposed by Yang *et al.* in [8] which also uses word embeddings, and the approach proposed in [7] which embeds bug reports into character level n-grams. For DWEN, we optimize on the number of dimensions for the word embedding approach and, number of layers and the batch size for deep neural network. For the word embedding approach, we used the Skipgram model since, it was used in [8] with the optimized size of dimensions as 300 and we observed the best results for deep neural network with 2 hidden layers and a batch size of 2^9 (512).

Results: We compare Recall Rate which has been a well accepted metric for evaluation of duplicate bug report detection task [7]. *Recall rate (RR) measures the accuracy of bug similarity computing systems in terms of counting the percentage of duplicates (a query which is a duplicate) for which the master bug-report is found within the top-N search results.* Table 1 reports the Recall Rate till 1, 5 and 20. It can be seen that the proposed approach, *DWEN* performs the best amongst the compared approaches. We believe the reason for



(a) DWEN (b) Yang *et al.* [8]

Figure 3: Probability distributions of duplicate bug report pairs and non-duplicate bug report pairs for Firefox project

higher performance of the proposed approach is due to the fact that the word embedding component in proposed approach is able to capture syntactic and semantic aspects of the word distribution in the bug reports database and deep neural network is able to capture how these word embedding features are related amongst the duplicate and non-duplicate reports. Yang *et al.* [8] also proposed an approach using word embeddings but it performs lower compared to the proposed approach, DWEN possibly because of the noise that is added due to the meta-features product and component information. In order to investigate further, we plot the probability distribution of duplicate and non-duplicate bug reports from Mozilla project for the proposed approach, DWEN and Yang *et al.*'s study which are shown in Figure 3. It can be seen from the graphs that the proposed approach able to learn the distinction of duplicate and non-duplicate bug reports as the mass for non-duplicate reports is mostly towards the left (in blue) and the mass for duplicates is at the right (in red). However, for Yang *et al.*'s approach, it can be seen the distribution for duplicate reports is distributed throughout the graph instead of being around higher probability side.

3 CONCLUSIONS AND FUTURE DIRECTIONS

In this initial work, we investigate an approach based on word embeddings and deep learning to compute bug report similarity. We show that the proposed approach is able to perform better than baselines and related approaches. As a part of future work, we aim at investigating different word embedding models for DWEN. We also aim to reduce the training to a single step compared the two step training process of training embeddings and deep neural network.

REFERENCES

[1] David M Blei et al. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research* (2003).
[2] Nathan Klein et al. 2014. New features for duplicate bug detection. In *Mining Software Repositories*. ACM.
[3] Alina Lazar et al. 2014. Generating duplicate bug datasets. In *Mining Software Repositories*. ACM.
[4] Tomas Mikolov et al. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
[5] Chengnian Sun et al. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *International Conference on Software Engineering*. ACM.
[6] Chengnian Sun et al. 2011. Towards more accurate retrieval of duplicate bug reports. In *Automated Software Engineering*. IEEE Press.
[7] Ashish Sureka et al. 2010. Detecting duplicate bug report using character n-gram-based features. In *Asia Pacific Software Engineering Conference*. IEEE Press.
[8] Xinli Yang et al. 2016. Combining word embedding with information retrieval to recommend similar bug reports. In *International Symposium on Software Reliability Engineering*. IEEE.
[9] Hugo Zaragoza et al. 2004. Microsoft Cambridge at TREC 13.. In *TREC*.