DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging

Senthil Mani, Anush Sankaran, Rahul Aralikatte
IBM Research, India
{sentmani,anussank,rahul.a.r}@in.ibm.com

ABSTRACT
For a given software bug report, identifying an appropriate developer who could potentially fix the bug is the primary task of a bug triaging process. A bug title (summary) and a detailed description is present in most of the bug tracking systems. Automatic bug triaging algorithm can be formulated as a classification problem, which takes the bug title and description as the input, mapping it to one of the available developers (class labels). The major challenge is that the bug description usually contains a combination of free unstructured text, code snippets, and stack trace making the input data highly noisy. The existing bag-of-words (BOW) feature models do not consider the syntactical and sequential word information available in the unstructured text.

In this research, we propose a novel bug report representation algorithm using an attention based deep bidirectional recurrent neural network (DBRNN-A) model that learns a syntactic and semantic feature from long word sequences in an unsupervised manner. Instead of BOW features, the DBRNN-A based bug representation is then used for training the classifier. Using an attention mechanism enables the model to learn the context representation over a long word sequence, as in a bug report. To provide a large amount of data to learn the feature learning model, the unixed bug reports (constitute about 70% bugs in an open source bug tracking system) are leveraged upon as an important contribution of this research, which were completely ignored in the previous studies. Another major contribution is to make this research reproducible by making the source code available and creating a public benchmark dataset of bug reports from three open source bug tracking system: Google Chromium, Mozilla Core, and Mozilla Firefox. For our experiments, we use 383,104 bug reports from Google Chromium, 314,388 bug reports from Mozilla Core, and 162,307 bug reports from Mozilla Firefox. Experimentally we compare our approach with BOW model and softmax classifier, support vector machine, naive Bayes, and co-

1 INTRODUCTION
In an usual process, the end user encounters a bug (also called an issue or a defect) while working on the system and reports the issue in a bug tracking system [8]. Fig 1 shows a sample screenshot of a bug reported in Google Chromium project (bug ID: 638277). The bug report usually consists of a brief summary and a detailed description at the time of reporting.

developers who could potentially solve the bug. Thus, in the rest of this research bug triaging refers to the task of developer assignment for the bug report [1]. In large scale systems, with a huge amount of incoming bugs, manually analyzing and triaging a bug report is a laborious process. Manual bug triaging is usually performed using the bug report content, primarily consisting of the summary and description. While additional sources of input has been explored in the literature such as developer profiling from github [3] and using component information [5], majority of the research efforts have focused on leveraging the bug report content for triaging [2] [14] [27] [28] [29] [32] [33]. Using the bug report content, automated bug triaging can be formulated as a classification problem, mapping the bug title and description to one of the developers (class labels). However, the bug report content contains noisy text information including code snippets, and stack trace details, as observed in Fig. 1. Processing such unstructured and noisy text data is a major challenge in learning a classifier.

Labeled bug report: 599892
Fixed by: brettw@chromium.org

Title: GN should only load each import once

Description: In GN multiple BUILD files can load the same import. GN caches the results of imports so we don’t have to load them more than once. But if two BUILD files load the same import at the same time, there is a race. Rather than lock, the code allows each to load the file and the first one finishes “wins”. This is based on the theory that the race is rare and processing imports is relatively fast. On Windows, many build files end up with the visual_studio_version.gni file which ends up calling build\vs_toolchain.py. This script can be quite slow (slower than the rest of the entire GN run in some cases). The result is that the race is guaranteed to happen for basically every BUILD file that references the .gni file, and we end up running the script many times in parallel (which only slows it down more). We should add the extra locking to resolve the race before loading rather than after.

Figure 1: Screenshot of a bug report available in Google Chromium project, bug ID: 638277. The bug report usually consists of a brief summary and a detailed description at the time of reporting.

Figure 2: A bug report from Google Chromium bug repository used as a labeled template for training the classifier.
1.1 Motivating Example
Consider a labeled bug report example shown in Fig. 2. The bag-of-words (BOW) feature representation of the bug report creates a boolean array marking true (or term-frequency) for each vocabulary word in the bug report [2]. During training, a classifier will learn this representation to the corresponding class label brettw@chromium.org. For the given two testing examples shown in Fig. 3, the actual fixer of first example, with bug id 634446, is brettw@chromium.org while the second example bug with id 616034 was fixed by machenb@chromium.org. However, based on BOW features there are 12 words common between testing report#1 and the train report, while there are 21 words common between testing report#2 and the train report. Hence, a BOW model mis-classifies that the testing bug report#2 with id 616034 should be fixed by brettw@chromium.org. The reasons for the mis-classification are:
(i) BOW feature model considers the sentence as a bag-of-words loosing the ordering (context) of words, and (ii) the semantic similarity between synonymous words in the sentence are not considered. Even though a bag-of-n-grams model considers a small context of word ordering, they suffer from high dimensionality and sparse data [12]. The semantic similarity between word tokens can be learnt using a skip-gram based neural network model called word2vec [22]. This model relies on distributional hypothesis which claims that words that appear in the same context in the sentence share a semantic meaning. Ye et al., [34] built a shared word representation using word2vec for word tokens present in code language and word tokens present in descriptive language. The main disadvantage of word2vec is that it learns a semantic representation of individual word tokens, however, does not consider a sequence of word tokens such as a sentence. An extension of word2vec called paragraph vector [19] considers the ordering of words, but only for a small context.

The rest of the paper is organized as follows: section 2 highlights the main research questions addressed in this research work and the key contributions, section 3 details the proposed approach including the deep learning algorithm and the classifier, section 4 talks about the experimental data collected in this research, section 5 discuss our experimental results and analysis, section 6 discusses some of the threats to validate our claims, section 7 talks about other applications that can be addressed using the proposed feature learning algorithm, section 8 explains about some closely related work and section 9 concludes our work with some future directions.

2 RESEARCH CONTRIBUTIONS
Learning semantic representation from large pieces of text (such as in description of bug reports), preserving the order of words, is a challenging research problem. Thus, we propose a deep learning technique, which will learn a succinct fixed-length representation of a bug report content in an unsupervised fashion i.e., the representation will be learnt directly using the data without the need for manual feature engineering. The main research questions (RQ) that we are trying address in this research paper are as follows:

(1) RQ1: Is it feasible to perform automated bug triaging using deep learning?

(2) RQ2: How does the unsupervised feature engineering approach perform, compared to traditional feature engineering approaches?

(3) RQ3: Is there an effect on the number of training samples per class on the performance of the classifier?

(4) RQ4: What is the effect of using only the title (or summary) of the bug report in performing triaging compared with the using the description as well?

(5) RQ5: Is transfer learning effective using deep learning, where the deep learning model is trained using one dataset and used to perform triaging in another dataset?

Recently, recurrent neural network (RNN) based deep learning algorithms have revolutionized the concept of word sequence representation and have shown promising breakthroughs in many applications such as language modeling and machine translation. Lam et al. [17] used deep neural network (DNN) with rSVM to learn a common representation between source code and the bug reports and used it for effective bug localization. White et al., [30] provided a broad perspective on how deep learning can be used in software repositories to solve some challenging problems. The main contributions of this research are summarized as follows:

- A novel bug report representation approach is proposed using DBRNN-A: Deep Bidirectional Recurrent Neural Network with Attention mechanism and with Long Short-Term Memory units (LSTM) [24]. The proposed deep algorithm algorithm is capable of remembering the context over a long sequence of words.

Figure 3: Two example bug reports from Google Chromium bug repository for which a suitable developer has to be predicted. By ground truth, bug report #1 was fixed by the same developer as in the training instance. However, the BOW feature of bug report #2 is more similar to the training instance than the bug report #1. The overlapping words with the training bug are highlighted.

<table>
<thead>
<tr>
<th>Bug report #1 to be triaged: 634446</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed by: <a href="mailto:brettw@chromium.org">brettw@chromium.org</a></td>
</tr>
<tr>
<td>Title: GN toolchain_args should be a scope rather than a function</td>
</tr>
<tr>
<td>Description: Currently in a toolchain_args overrides are:</td>
</tr>
<tr>
<td>toolchain_args() { foo = 1 bar = ”baz” }</td>
</tr>
<tr>
<td>We’re transitioning this to be a scope type:</td>
</tr>
<tr>
<td>toolchain_args = { foo = 1 bar = ”baz” }</td>
</tr>
<tr>
<td>which will allow the gcc_toolchain template to forward values from the invoker without it having to know about all build args ever overridden in the entire build.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bug report #2 to be triaged: 616034</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed by: <a href="mailto:machenb@chromium.org">machenb@chromium.org</a></td>
</tr>
<tr>
<td>Title: GN toolchain_args should be a scope rather than a function</td>
</tr>
<tr>
<td>Description: Can v8_use_external_startup_data be overridden in a chromium build? On the one hand, there is the default, declared as a gn arg, which is true. On the other hand, there is “v8_use_external_startup_data = !is_ios” as a build override in chromium. There is no logic to not override if the user changes the gn arg. The same would hold for v8_optimized_debug.</td>
</tr>
<tr>
<td>This would mean that the declared arg cannot be overwritten via command line.</td>
</tr>
</tbody>
</table>
The untriaged and unsolved bug reports constitute about 70% in an open source bug repository and are usually ignored in the literature [14]. In this research, we provide a mechanism to leverage all the untriaged bugs to learn bug representation model in an unsupervised manner.

Experimental data (bug reports) are collected from three open source bug repositories: 3, 83, 104 from Chromium, 3, 14, 388 from Mozilla Core, and 1, 62, 307 from Mozilla Firefox. Performance of the classifiers trained on different train-test splits of datasets [14] [18] are neither comparable nor reproducible. Thus, to enable our research reproducible, the entire dataset along with the exact train test split and the source code of our approach are made publicly available for research purpose.

We further study the effectiveness of the proposed bug training in a cross-data testing scenario (transfer learning). By training the model with bugs from Chromium project and re-using the model for triaging bugs in Core and Firefox projects (Mozilla bug repository), the transfer learning ability of the deep learning model is showcased.

3 PROPOSED APPROACH

The problem of automated bug triaging of software bug reports is formulated as a supervised classification approach with the input data being the bug summary (title) and the bug description. Fig. 4 highlights the major steps involved the proposed automated bug triaging algorithm and are explained as follows:

1. a bug corpus having title, description, reported time, status, and owner is extracted from an open source bug tracking system,
2. handling the URLs, stack trace, hex code, and the code snippets in the unstructured description requires specialized training of the deep learning model, and hence in this research work, those contents are removed in the preprocessing stage,
3. a set of unique words that occurred for at least k-times in the corpus is extracted as the vocabulary,
4. the triaged bugs (D2) are used for classifier training and test, while all the untriaged/open bugs (D1) are used to learn a deep learning model,
5. a deep bidirectional recurrent neural network with attention mechanism technique learns a bug representation considering the combined bug title and description as a sequence of word tokens,
6. the triaged bugs (D2) are split into train and test data with a 10 fold cross validation to remove training bias,
7. feature representation for the training bug reports are extracted using the learnt DB-RNN algorithm,
8. a supervised classifier is trained for performing developer assignment as a part of bug triaging process,
9. feature representation of the testing bugs are then extracted using the learnt deep learning algorithm,
10. using the extracted features and the learnt classifier, a probability score for every potential developer is predicted and the accuracy is computed in the test set.

The proposed approach varies with the traditional pipeline for automated bug triaging in (i) step 4 where the untriaged bugs (D1) are completely ignored and (ii) the deep learning based bug report representation instead of a bag-of-words representation. Addition of steps 4 and 5, enables to automatically learn a bug report representation from the data itself instead of manual engineering.

3.1 Deep Bidirectional Recurrent Neural Network with Attention (DBRNN-A)

The proposed DBRNN-A based feature learning approach, as shown in Fig. 5, have the following key advantages:

- DBRNN-A can learn sentence representation preserving the order and syntax of words, as well as, retaining the semantic relationship. Long short-term memory (LSTM) cells [13] are used in the hidden layer which have a memory unit that can remember longer word sequences and also can solve the vanishing gradient problem [25].
- Intuitively, all the words in the content may not be useful in triaging the bug. To incorporate this, we introduce an attention mechanism [20] which learns to ‘attend’ to only the important words in the bug report during classification.
As the attention mechanism chooses only a few words during classification, the DBRNN-A can learn context representation from really long word sequences.

- A bidirectional RNN [9] considers the word sequence both in forward direction (first word to last word) and in backward direction (last word to first word) and merges both these representations. Thus, a context of a particular word includes both the previous few words and following few words making the representation more robust.

For each word, a one-hot $|V|$-dimensional representation is extracted using the vocabulary $^2$, over which a a $|P|$-dimensional word2vec representation [23] is learnt. As shown in Fig. 5 (a), a DBRNN-A with LSTM units is learnt over this word representation, to obtain a $|D|$-dimensional feature representation of the entire bug report (title + description). RNN is a sequence network containing a hidden layer with $m$ hidden units, $h = \{h_1, h_2, \ldots, h_m\}$. The input to the system is a sequence of word representations, $x = \{x_1, x_2, \ldots, x_m\}$, and produces a sequence of outputs $y = \{y_1, y_2, \ldots, y_m\}$. Each hidden unit is a state model converting the previous state, $s_{i-1}$, and a word, $x_i$ to the next state, $s_i$ and an output word, $y_i$. The term “recurrent” explains that every hidden unit performs the same function in recurrent fashion, $f : \{s_{i-1}, x_i\} \rightarrow \{s_i, y_i\}$. Intuitively, the state $s_i$ carries the cumulative information of the $i$ previous words observed. The output $y_m$ obtained from the last hidden node is a cumulative representation of the entire sentence. For example, consider the tokenized input sentence provided in Fig. 5. When $i = 1$, $x_1$ is the $|P|$-dimensional word2vec representation of the input word, unresponsive and the previous state $s_0$ is randomly initialized. Using the LSTM function $f$, the current state $s_1$ and the word output $y_1$ are predicted. Given the next word stop and the current state $s_1$, the same function $f$ is used to predict $s_2$ and $y_2$. The shared function reduces the number of learnable parameters as well as retains the context from all the words in the sequence. For language modeling or learning sentence representation, the ground truth $y_i$ is the next word in the sequence $x_{i+1}$, that is, upon memorizing the previous words in the sentence the network tries to predict the next word. LSTM function [9] have special purpose built-in memory units to store the context information over longer sentences.

Further, to selectively remember and learn from the important words in a bug report, an attention model is employed. An attention vector is derived by performing a weighted summation of all the computed outputs, $y_i$, as follows:

$$a_m = \sum_{i=1}^{m} \alpha_i y_i \tag{1}$$

Intuitively, $\alpha_i$ associates a weight to each word implying the importance of that word for classification. Two different deep RNN based feature model are learnt, one with input word sequence running forward and one with input word sequence running backward. The final representation, $r$, obtained for a bug report, is provided as follows:

$$r = \sum_{m} \alpha_m \oplus \sum_{m} \alpha_m \oplus a_m \tag{2}$$

where $\oplus$ represents concatenation of the vectors. In comparison as shown in Fig. 5 (b), a term frequency based BOW model would produce a $|V|$-dimensional representation for the same bug report, where $V$ is the size of vocabulary. Typically, the size of $|P|$ is chosen as 300 [23] and the size of $D$ will be less than $4|P| (< 1200)$ is much smaller than the size of $|V|$. For example, consider 10,000 bugs used for training with 250,000 unique words ($|V|$). BOW model representation would produce a sparse feature matrix of

---

A huge corpus of bug report data is obtained from three popular open source systems: Chromium, Mozilla Core, and Mozilla Firefox used in our experiments.

The entire deep learning model was implemented in Python using Keras library. To the best of our knowledge, this is the first time a deep sequence learning model has been applied to learn a bug representation and used those features to learn a supervised model for automated software bug triaging.

### 3.2 Classifying (Triaging) a Bug Report

The aim of the supervised classifier is to learn a function, $C$, that maps a bug feature representation to a set of appropriate developers. Formulating automated bug triaging as a supervised classification problem has been well established in literature [5] [32]. However, it is well understood that a classification is only as good as the quality of features. Hence, the major contribution in this research is to propose a better bug report representation model and to improve the performance of existing classifiers. In this research, we use a softmax classifier, a popular choice of classifier along with deep learning [9] [7] [10]. Softmax classifier is a generalization of logistic regression for multiclass classification, taking the features and providing a vector of scores with length equal to the number of the classes. A softmax classifier normalizes these score values and provides an interpretable probability value of the $i$-th bug report belonging to the class.

### 4 LARGE SCALE PUBLIC BUG TRIAGE DATASET

A huge corpus of bug report data is obtained from three popular open source systems: Chromium, Mozilla Core, and Mozilla Firefox and the data collection process is explained in this section.

To make this research reproducible, the entire data along with the exact train-test protocol and with source code is made available at: http://bugtriage.mybluemix.net/.

#### 4.1 Data Extraction

Bug reports from the Google Chromium project were downloaded for the duration of August 2008 (Bug ID: 2) - July 2016 (Bug ID: 633012). A total of 383,104 bugs where collected with the bug title, description, the bug owner, and the reported time. The developer in the “owner” field is considered as the ground truth triage class for the given bug. Bugs with status as Verified or Fixed, and type as bug, and has a valid ground truth bug owner are used for training and testing the classifier while rest of the bugs are used for learning a bug representation. However, we noticed that there were a total of 11,044 bug reports with status as Verified or Fixed and did not have a valid owner associated. These bugs are considered as open bugs, resulting in a total of 263,936 (68.9%) bug reports are used for deep learning, and 118,643 (31%) bugs are used for the classifier training and testing.

Data from two popular components from Mozilla bug repository are extracted: Core and Firefox. 314,388 bug reports are extracted from Mozilla Core reported between April 1998 (Bug ID: 91) and June 2016 (Bug ID: 1278040), and 162,307 bug reports are extracted from Mozilla Firefox reported between July 1999 (Bug ID: 10954) and June 2016 (Bug ID: 1278030). The developer in the “Assigned To” is considered as the ground truth triage class during classification.

For training and testing the classifier while rest of the bugs are used for learning a bug representation. However, we noticed that there were a total of 11,044 bug reports with status as Verified or Fixed and did not have a valid owner associated. These bugs are considered as open bugs, resulting in a total of 263,936 (68.9%) bug reports are used for deep learning, and 118,643 (31%) bugs are used for the classifier training and testing.

Table 1: Summary of the three different bug repositories, Google Chromium, Mozilla Core, and Mozilla Firefox used in our experiments.

<table>
<thead>
<tr>
<th>Property</th>
<th>Chromium</th>
<th>Core</th>
<th>Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total bugs</td>
<td>383,104</td>
<td>314,388</td>
<td>162,307</td>
</tr>
<tr>
<td>Bugs for learning feature</td>
<td>263,936</td>
<td>186,173</td>
<td>138,093</td>
</tr>
<tr>
<td>Bugs for classifier</td>
<td>118,643</td>
<td>128,215</td>
<td>24,214</td>
</tr>
<tr>
<td>Vocabulary size $</td>
<td>71,575</td>
<td>122,578</td>
<td>57,922</td>
</tr>
</tbody>
</table>

Data from two popular components from Mozilla bug repository are extracted: Core and Firefox. 314,388 bug reports are extracted from Mozilla Core reported between April 1998 (Bug ID: 91) and June 2016 (Bug ID: 1278040), and 162,307 bug reports are extracted from Mozilla Firefox reported between July 1999 (Bug ID: 10954) and June 2016 (Bug ID: 1278030). The developer in the “Assigned To” is considered as the ground truth triage class during classification. Bug reports with status as verified fixed, resolved fixed, and closed fixed are used for classifier training and testing. However, some of the fixed reports did not have a developer assigned to it, such as, in Core (7219/135434 = 5.33%) and in Firefox (3716/27930 = 13.3%). After ignoring these bugs, a final number of 1,28,215 bugs for Core and 24,214 bugs for Firefox are considered for classifier training and testing. The summary of the datasets is provided in Table 1.

#### 4.2 Data Preprocessing

The three datasets are preprocessed independently using the same set of steps and a benchmark protocol is created. For every bug report, the title and description text content of the bug are combined. Preprocessing of the unstructured textual content involves removing URLs, hex code, and stack trace information, and converting all text to lower case letters. Tokenization of words is performed using Stanford’s NLTK package. A vocabulary of all words is constructed using the entire corpus. To remove rarely occurring words and reduce the vocabulary size, usually the top-F frequent words are considered or only those words occurring with a minimum frequency are considered [34]. For the extracted data, we experimentally observed that a minimum word frequency of 5 provided a good trade-off between the vocabulary size and performance.

#### 4.3 Training Data for Deep Learning

In our data split mechanism, the classifier testing data is unseen data and hence cannot be used for the deep learning algorithm. A design choice was taken for not using the classifier training data for training the deep learning model, as including them only marginally improved the accuracy but largely increased the training time. Thus, only the untriaged bugs (explained in the data extraction subsection) is used for training the deep learning model. Also, using a non-overlapping dataset for training the feature model and training the classifier model highlights the generalization ability of the features.

#### 4.4 Training Data for Classification

For training and testing the supervised classifier, a 10-fold cross validation model as proposed by Bettenburg et al [4] is followed. All the fixed bug reports are arranged in chronological order and split into 11 sets. Starting from the second fold, every fold is used as a test set, with the cumulation of previous folds for training.

---

1. https://bugs.chromium.org/p/chromium/issues/list
2. https://bugzilla.mozilla.org/
Typically, in an open source project the developers keep changing overtime, and hence chronological splitting ensures that the train and test sets have highly overlapping developers. Further, in order to make the training effective, we need more number of training sample per developer. In a recent study, Jonsson et al., [14] trained using those developers who have at least addressed 50 bug reports i.e., minimum number of training samples per class is 50. From different studies in the literature [2] [5], it is clear that the threshold parameter affect the classification performance. Thus, in this research we study the direct relation between the threshold value and the classification performance, by having four different thresholds for the minimum number of training samples per class as 0, 5, 10, 20. To perform a closed training experiment, it is made sure that all the classes available in training are available for training while there are additional classes in training which are not available in the test set. Thus, for every test bug report with an owner, the classifier is already trained with other bugs trained by the same owner.

5 EXPERIMENTAL EVALUATION

5.1 Evaluation Protocol and Metric

For a given bug report, the trained softmax classifier provides a probability value for every developer, denoting their association with the bug report. Thus, the evaluation metric that is used is the top-k accuracy, which denotes the ratio of the bug reports for which the actual developer is present in the top-k retrieved results. Across the cross validation (CV) sets, varying classes or a set of developers are used. Thus during CV#1, the classes used for training and testing is different from the classes used in CV#2. Thus, as the classifier model across the CV is trained on different classes, taking the average accuracy would only provide a ballpark number of the performance, while is not accurately interpretable. Thus, it is required to report the top-k accuracy of each cross validation set to understand the variance introduced in the model training [16].

For learning the deep representation, a DBRNN-A is constructed having 300 LSTM units and the dropout probability is 0.3. A categorical cross entropy based loss function is used with Adam optimizer, learning rate as 0.001, and trained for 100 epochs with early stopping. The model architecture and parameters utilized are shown in Fig. 6.

5.2 Comparison with Existing Algorithms

The major challenge in cross comparison of algorithm performance is the lack of a public benchmark dataset and source code implementation of the existing research. Thus, the bug triaging accuracy obtained in the previous research works cannot be compared with the proposed approach, unless the results are shown in the same dataset. Thus, we implement some of the successful approaches for automated bug triaging from the literature [2] [32] [14] and compare it with our algorithm using our benchmark dataset. Term frequency based BOW is used to represent the combined title and description from a bug report, as shown in Fig. 5. Using these features, we evaluate the performance of four different classifiers: (i) Softmax classifier [26], (ii) Support Vector Machine (SVM) [31], (iii) Multinomial Naive Bayes (MNB) [15], and (iv) Cosine distance based matching [21]. The four supervised classifiers are implemented using the Python scikit-learn\(^7\) package. All these four classifiers use only the classifier training and testing data and do not use the untriaged bug reports.

\(^7\)http://scikit-learn.org/
### Table 3: Rank-10 accuracy obtained on the Mozilla Core project across the ten cross validations. The average accuracy over the cross validation and standard deviation is also reported. The best performing results are shown in bold.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Classifier</th>
<th>CV#1</th>
<th>CV#2</th>
<th>CV#3</th>
<th>CV#4</th>
<th>CV#5</th>
<th>CV#6</th>
<th>CV#7</th>
<th>CV#8</th>
<th>CV#9</th>
<th>CV#10</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. train samples per class = 0</td>
<td>BOW + MNB</td>
<td>21.6</td>
<td>23.6</td>
<td>29.7</td>
<td>30.3</td>
<td>31.0</td>
<td>31.2</td>
<td>31.9</td>
<td>31.7</td>
<td>31.7</td>
<td>32.3</td>
<td>32.1</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>16.3</td>
<td>17.4</td>
<td>19.5</td>
<td>21.3</td>
<td>22.5</td>
<td>23.2</td>
<td>24.0</td>
<td>25.5</td>
<td>27.5</td>
<td>29.1</td>
<td>22.6</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>13.6</td>
<td>14.6</td>
<td>14.9</td>
<td>14.0</td>
<td>12.1</td>
<td>12.9</td>
<td>11.7</td>
<td>13.7</td>
<td>14.4</td>
<td>14.1</td>
<td>13.6</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>14.3</td>
<td>11.8</td>
<td>9.5</td>
<td>10.0</td>
<td>9.2</td>
<td>10.4</td>
<td>10.5</td>
<td>10.6</td>
<td>11.0</td>
<td>10.8</td>
<td>10.4</td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>30.1</td>
<td>31.7</td>
<td>35.2</td>
<td>33.0</td>
<td>34.1</td>
<td>35.9</td>
<td>34.8</td>
<td>34.2</td>
<td>34.6</td>
<td>33.1</td>
<td><strong>33.9 ± 1.7</strong></td>
</tr>
<tr>
<td>Min. train samples per class = 5</td>
<td>BOW + MNB</td>
<td>20.7</td>
<td>23.8</td>
<td>29.7</td>
<td>31.4</td>
<td>31.7</td>
<td>33.8</td>
<td>35.6</td>
<td>36.7</td>
<td>35.8</td>
<td>36.2</td>
<td>35.2</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>15.7</td>
<td>17.7</td>
<td>19.9</td>
<td>21.4</td>
<td>22.8</td>
<td>24.7</td>
<td>26.4</td>
<td>27.5</td>
<td>29.4</td>
<td>29.9</td>
<td>23.5</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>16.4</td>
<td>12.9</td>
<td>11.5</td>
<td>10.4</td>
<td>13.4</td>
<td>13.8</td>
<td>12.7</td>
<td>12.0</td>
<td>12.8</td>
<td>13.1</td>
<td>12.9</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>14.9</td>
<td>13.5</td>
<td>12.5</td>
<td>10.6</td>
<td>11.4</td>
<td>12.8</td>
<td>12.1</td>
<td>13.3</td>
<td>12.4</td>
<td>14.0</td>
<td>12.7</td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>33.8</td>
<td>31.5</td>
<td>35.8</td>
<td>35.3</td>
<td>34.7</td>
<td>36.8</td>
<td>37.1</td>
<td>38.4</td>
<td>37.7</td>
<td>38.0</td>
<td><strong>35.9 ± 2.1</strong></td>
</tr>
<tr>
<td>Min. train samples per class = 10</td>
<td>BOW + MNB</td>
<td>18.4</td>
<td>23.9</td>
<td>29.8</td>
<td>33.4</td>
<td>36.7</td>
<td>39.4</td>
<td>38.5</td>
<td>40.8</td>
<td>41.3</td>
<td>42.5</td>
<td>34.5</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>16.0</td>
<td>18.0</td>
<td>20.0</td>
<td>21.4</td>
<td>22.7</td>
<td>25.7</td>
<td>27.8</td>
<td>30.4</td>
<td>33.1</td>
<td>35.5</td>
<td>25.1</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>17.5</td>
<td>15.6</td>
<td>16.5</td>
<td>16.4</td>
<td>16.4</td>
<td>17.0</td>
<td>17.2</td>
<td>17.4</td>
<td>16.9</td>
<td>16.2</td>
<td>16.7</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>15.6</td>
<td>14.2</td>
<td>14.4</td>
<td>13.9</td>
<td>14.0</td>
<td>13.4</td>
<td>13.8</td>
<td>14.5</td>
<td>14.9</td>
<td>14.1</td>
<td>14.3</td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>32.5</td>
<td>33.7</td>
<td>35.5</td>
<td>36.5</td>
<td>36.4</td>
<td>34.4</td>
<td>36.1</td>
<td>37.3</td>
<td>38.9</td>
<td>39.6</td>
<td><strong>36.1 ± 2.1</strong></td>
</tr>
<tr>
<td>Min. train samples per class = 20</td>
<td>BOW + MNB</td>
<td>21.3</td>
<td>24.3</td>
<td>30.2</td>
<td>34.8</td>
<td>38.5</td>
<td>39.4</td>
<td>37.5</td>
<td>40.7</td>
<td>42.1</td>
<td>41.8</td>
<td>35.1</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>16.8</td>
<td>18.4</td>
<td>20.4</td>
<td>23.3</td>
<td>28.6</td>
<td>31.3</td>
<td>35.7</td>
<td>38.6</td>
<td>37.3</td>
<td>38.9</td>
<td>28.9</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>14.6</td>
<td>15.2</td>
<td>16.4</td>
<td>14.5</td>
<td>13.9</td>
<td>15.7</td>
<td>16.8</td>
<td>15.6</td>
<td>16.1</td>
<td>16.4</td>
<td>15.5</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>18.8</td>
<td>16.4</td>
<td>11.4</td>
<td>10.5</td>
<td>11.8</td>
<td>13.1</td>
<td>13.6</td>
<td>14.3</td>
<td>14.8</td>
<td>15.3</td>
<td><strong>14.0 ± 2.4</strong></td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>33.3</td>
<td>34.9</td>
<td>36.5</td>
<td>36.8</td>
<td>37.7</td>
<td>39.0</td>
<td>41.3</td>
<td>42.6</td>
<td>41.1</td>
<td>43.3</td>
<td><strong>38.8 ± 3.2</strong></td>
</tr>
</tbody>
</table>

### 5.3 Result Analysis

The results obtained in the Google Chromium, Mozilla Core, and Mozilla Firefox datasets are shown in Table 2, Table 3, and Table 4, respectively. The main research questions focused in this paper are answered using the obtained results.

**RQ1:** Is it feasible to automated perform bug triaging using deep learning?
Learning the feature representation model and training the classifier are usually offline tasks and do not contribute towards the testing time. For example in the Google Chromium dataset, training the DBRNN-A takes about 300 seconds per epoch. For the entire CV#10 subset, training and testing time the softmax classifier takes about 121 seconds and 73 seconds, respectively. However, after training the models, developer assignment for a new bug report takes only 8 milliseconds using the proposed approach (feature extraction + classification), highlighting the speed of the proposed approach.

**RQ2: How does the unsupervised feature engineering approach perform, compared to traditional feature engineering approaches?**

It can be concretely observed from the results that the feature learning using DBRNN-A outperforms the traditional BOW feature model. In Chromium dataset, rank-10 average accuracy of BOW + Softmax is around 9 – 12%, while the best performing classifier provides 26 – 28%. This shows the challenging nature of the bug triaging problem in the large dataset that we have created. However, DBRNN-A provides a rank-10 average accuracy in the range of 37 – 43% improving results by 12 – 15%. Similarly in Mozilla Core, we observe a 3 – 5% improvement and in Mozilla Firefox, we observe a 7 – 17% improvement in rank-10 average accuracy by using deep learning features because the deep learning model could memorize the context from longer sentences in a bug report reasoning for the large improvement in performance. From the results, we observe that for BOW features MNB and cosine distance based matching outperforms SVM and softmax classifier. Although SVM is a popular choice of a supervised classifier, for real numbered sparse features in BOW model, feature independence which is assumed both in MNB and cosine distance matching proves successful.

![Figure 8: The rank-10 average accuracy of the deep learning algorithm on all three datasets by using only title or title along with the description in bug report. Discarding the description reduces the performance significantly, addressing RQ4.](image-url)

From the obtained results, it can be clearly observed that the deep learning the representation of a bug report is a feasible and potentially competent approach for bug triaging. The proposed DBRNN-A approach provided rank-10 triaging accuracy in the range of 34 – 47%. All the experiments are executed in an Intel(R) Xeon(R) CPU E5-2660 v3, running at 2.60GHz and with a Tesla K80 GPU.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Classifier</th>
<th>CV#1</th>
<th>CV#2</th>
<th>CV#3</th>
<th>CV#4</th>
<th>CV#5</th>
<th>CV#6</th>
<th>CV#7</th>
<th>CV#8</th>
<th>CV#9</th>
<th>CV#10</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. train samples per class = 0</td>
<td>BOW + MNB</td>
<td>19.1</td>
<td>21.3</td>
<td>24.5</td>
<td>22.9</td>
<td>25.8</td>
<td>28.1</td>
<td>30.3</td>
<td>31.9</td>
<td>33.94</td>
<td>35.55</td>
<td>27.4 ± 5.2</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>17.3</td>
<td>20.3</td>
<td>22.9</td>
<td>25.4</td>
<td>26.9</td>
<td>28.3</td>
<td>29.8</td>
<td>27.5</td>
<td>28.9</td>
<td>30.1</td>
<td>25.7 ± 4.1</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>13.4</td>
<td>11.4</td>
<td>13.8</td>
<td>15.5</td>
<td>14.5</td>
<td>14.5</td>
<td>14.3</td>
<td>14.4</td>
<td>14.6</td>
<td>14.6</td>
<td>14.1 ± 1.0</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>11.9</td>
<td>17.8</td>
<td>17.8</td>
<td>15.7</td>
<td>13.6</td>
<td>15.5</td>
<td>13.7</td>
<td>13.1</td>
<td>13.1</td>
<td>13.6</td>
<td>14.6 ± 1.9</td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>33.6</td>
<td>34.2</td>
<td>34.7</td>
<td>36.1</td>
<td>38.0</td>
<td>37.3</td>
<td>38.9</td>
<td>36.3</td>
<td>37.4</td>
<td>38.1</td>
<td>36.5 ± 1.7</td>
</tr>
<tr>
<td>Min. train samples per class = 5</td>
<td>BOW + MNB</td>
<td>21.1</td>
<td>26.8</td>
<td>31.1</td>
<td>33.4</td>
<td>36.5</td>
<td>36.0</td>
<td>37.6</td>
<td>36.9</td>
<td>34.9</td>
<td>36.5</td>
<td>33.1 ± 5.1</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>20.8</td>
<td>23.0</td>
<td>23.7</td>
<td>26.2</td>
<td>27.4</td>
<td>29.2</td>
<td>32.3</td>
<td>32.7</td>
<td>34.1</td>
<td>35.2</td>
<td>28.5 ± 4.8</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>14.4</td>
<td>16.0</td>
<td>17.8</td>
<td>17.8</td>
<td>17.8</td>
<td>16.3</td>
<td>16.7</td>
<td>16.7</td>
<td>15.2</td>
<td>16.5</td>
<td>1.1 ± 1.1</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>18.2</td>
<td>14.8</td>
<td>16.7</td>
<td>16.7</td>
<td>15.4</td>
<td>14.5</td>
<td>12.5</td>
<td>12.9</td>
<td>12.9</td>
<td>13.7</td>
<td>14.8 ± 1.8</td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>27.6</td>
<td>34.9</td>
<td>37.9</td>
<td>38.7</td>
<td>40.1</td>
<td>42.3</td>
<td>45.2</td>
<td>44.9</td>
<td>45.0</td>
<td>44.5</td>
<td>40.1 ± 5.3</td>
</tr>
<tr>
<td>Min. train samples per class = 10</td>
<td>BOW + MNB</td>
<td>21.7</td>
<td>27.6</td>
<td>32.1</td>
<td>34.8</td>
<td>37.7</td>
<td>34.6</td>
<td>32.6</td>
<td>34.7</td>
<td>36.7</td>
<td>38.5</td>
<td>33.1 ± 4.8</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>18.1</td>
<td>21.2</td>
<td>24.4</td>
<td>27.0</td>
<td>28.3</td>
<td>30.1</td>
<td>32.3</td>
<td>34.00</td>
<td>35.4</td>
<td>36.6</td>
<td>28.7 ± 5.8</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>09.9</td>
<td>09.9</td>
<td>11.8</td>
<td>11.8</td>
<td>11.8</td>
<td>12.8</td>
<td>12.9</td>
<td>12.9</td>
<td>12.9</td>
<td>12.8</td>
<td>11.9 ± 1.1</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>14.3</td>
<td>15.6</td>
<td>12.1</td>
<td>09.5</td>
<td>09.5</td>
<td>11.2</td>
<td>12.0</td>
<td>12.6</td>
<td>12.1</td>
<td>12.7</td>
<td>12.1 ± 1.8</td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>35.1</td>
<td>36.4</td>
<td>40.5</td>
<td>42.5</td>
<td>45.4</td>
<td>47.4</td>
<td>48.9</td>
<td>49.1</td>
<td>51.1</td>
<td>51.4</td>
<td>44.8 ± 5.6</td>
</tr>
<tr>
<td>Min. train samples per class = 20</td>
<td>BOW + MNB</td>
<td>22.0</td>
<td>22.8</td>
<td>23.6</td>
<td>26.3</td>
<td>29.2</td>
<td>32.3</td>
<td>34.4</td>
<td>36.4</td>
<td>38.6</td>
<td>38.4</td>
<td>30.4 ± 6.2</td>
</tr>
<tr>
<td></td>
<td>BOW + Cosine</td>
<td>18.4</td>
<td>21.9</td>
<td>25.1</td>
<td>27.5</td>
<td>29.1</td>
<td>31.4</td>
<td>33.8</td>
<td>35.9</td>
<td>36.7</td>
<td>38.3</td>
<td>29.8 ± 6.3</td>
</tr>
<tr>
<td></td>
<td>BOW + SVM</td>
<td>18.7</td>
<td>16.9</td>
<td>15.4</td>
<td>18.2</td>
<td>20.6</td>
<td>19.1</td>
<td>20.3</td>
<td>21.8</td>
<td>22.7</td>
<td>21.9</td>
<td>19.6 ± 2.2</td>
</tr>
<tr>
<td></td>
<td>BOW + Softmax</td>
<td>16.5</td>
<td>13.3</td>
<td>13.2</td>
<td>13.8</td>
<td>11.6</td>
<td>12.1</td>
<td>12.3</td>
<td>12.3</td>
<td>12.5</td>
<td>12.9</td>
<td>13.1 ± 1.3</td>
</tr>
<tr>
<td></td>
<td>DBRNN-A + Softmax</td>
<td>38.9</td>
<td>37.4</td>
<td>39.5</td>
<td>43.9</td>
<td>45.0</td>
<td>47.1</td>
<td>50.5</td>
<td>53.3</td>
<td>54.3</td>
<td>55.8</td>
<td>46.6 ± 6.4</td>
</tr>
</tbody>
</table>

Table 4: Rank-10 accuracy obtained on the Mozilla Firefox project across the ten cross validations. The average accuracy over the cross validation and standard deviation is also reported. The best performing results are shown in bold.
6 THREATS TO VALIDITY

There are certain threats to establish the validity of the proposed results. While the common threats to any learning based classification system are applicable, few of these threats are specific to the advocated problem, as follows:

(1) The results are shown using three open source bug repositories with different characteristics to ensure generalization.

However, commercial bug tracking systems may follow different patterns and hence our results may not be directly extensible to such repositories.

(2) Currently in our approach, only the bug report title and description are considered. While the experimental results show that these two unstructured text data are necessary, there could be other additional information required to sufficiently triage a bug report.

(3) For comparison purpose, we re-implemented some of the successful algorithms in the literature for bug triaging, as there is no publicly implementation available. Although we have implemented the algorithms true to the best of our understanding, there could be some minor deviations from the original implementation.

(4) Both during training and testing of a classifier, we assumed only one developer as the rightful owner of a bug report. However, based on patterns and history of bug reports that are solved, there could be more than one active developer in the project who could potentially address the bug.

7 OTHER APPLICATIONS

The bug representation is learnt directly from the data in an unsupervised fashion and the features are task independent. This gives us a flexibility to use these features to learn a supervised classifier for any task or application. We have discussed a few of other possible applications for the proposed feature representation.

- Which bug gets fixed: It is a challenging research problem that have been addressed in literature [11]. Using the same feature representation extracted in this research, a supervised or semi-supervised binary classifier can be trained to classify fixed bugs with non-fixed bugs.
- Bug-fix time prediction: As Bhattacharya et al. [6] discuss, a predictive model can be constructed using the proposed features to learn the time required to fix the bug.
- Reopen analysis: It provides an interesting insight from the maintenance perspective to study which bugs get reopened during its lifecycle. As discussed by Zimmermann et al. [35], characterizing these bugs and predicting them can be performed using the deep learning features.
- Bug priority estimation: Priority of the bug is to be estimated before triaging happens [6]. Based on the priority, the SLA clock for the bug and the developer to be assigned might change. A 5-point scale priority can be formulated as a five class classification and using the learnt features, a supervised classifier can be learnt.

8 RELATED WORK

Table 5 presents a list of closely related works on bug triaging arranged in a chronological order (year 2010 to 2016). A majority of previous techniques have used bug summary/title and description [2] [28] [32] [33] because they are available at the time of ticket submission and do not change in tickets’ lifecycle. Bhattacharya et. al. [5] use additional attributes such as product, component, and the last developer activity to shortlist developers. Shokripour et al. [27] use code information for improved performance. Badashian
et. al. [3] identify developers’ expertise using Stack Overflow and keywords from bug description.

From Table 5, we observe that many different feature models such as tf, normalized tf, tf-idf, and n-grams have been employed. Choosing which feature model to use is an engineering design choice and it is challenging to choose which feature model will best represent the collected data. In this research, we address this challenge and design a deep bidirectional RNN which directly learns the best feature representation from the data in an unsupervised fashion. Further, we move beyond a word level representation model and propose a sequence of word representation model, to learn a unified representation for the entire bug report.

### 9 CONCLUSION

In this research we proposed a novel software bug report (title + description) representation algorithm using deep bidirectional Recurrent Neural Network with attention (DBRNN-A). The proposed deep learning algorithm learns a paragraph level representation preserving the ordering of words over a longer context and also the semantic relationship. The performance of four different classifiers, multinomial naive Bayes, cosine distance, support vector machines, and softmax classifier are compared. To perform experimental analysis, bug reports from three popular open source bug repositories are collected - Google Chromium (383,104), Mozilla Core (314,388), and Mozilla Firefox (162,307). Experimental results show that DBRNN-A along with softmax classifier outperforms the bag-of-words model, improving the rank-10 average accuracy in all three datasets. Further, it was shown that using only the title information for triaging significantly reduces the classification performance highlighting the importance of description. The transfer learning ability of the deep learning model is experimentally shown, where the model learnt on the Chromium dataset competitively triaged the bugs in the Mozilla dataset. Additionally, the dataset along with its complete benchmarking protocol and the implemented source code is made publicly available to increase the reproducibility of this research.

### REFERENCES


DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging

Website: http://bugtriage.mybluemix.net/


