2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

Fast Detection of Duplicate Bug Reports using LDA-based Topic Modeling and Classification

Thangarajah Akilan, *M*ember, IEEE, Dhruvit Shah, Nishi Patel, Rinkal Mehta

***Abstract*—A bug tracking system continuously monitors the status of a software environment, like an Operating System (OS) or a user application. Whenever it detects an anomaly situation, it generates a bug report and sends it to the software developer or maintenance center. However, the newly reported bug can be an already existing issue that was reported earlier and may have a solution in the master report repository. This condition brings an avalanche of duplicate bug reports, posing a big challenge to the software development life cycle. Thus, early detection of duplicate bug reports has become an extremely important task in the software industry. To address this issue, this work proposes a double-tier approach using clustering and classification, whereby it exploits Latent Dirichlet Allocation (LDA) for topic-based clustering, multimodal text representation using Word2Vec (W2V), FastText (FT) and Global Vectors for Word Representation (GloVe), and a unified text similarity measure using Cosine and Euclidean metrics. The proposed model is tested on the Eclipse dataset consisting over 80,000 bug reports, which is the amalgamation of both master and duplicate reports. This work considers only the description of the reports for detecting duplicates. The experimental results show that the proposed two-tier model achieves a recall rate of 67% for Top-N recommendations with 3 times faster computation than the conventional one-on-one classification model.**

***Index Terms*—Bug report detection, topic modeling, information retrieval, natural language processing, machine learning**

I. Introduction

During the process of development, maintenance and utilization of a software application, there are consistently arising issues such that the functionality of the whole system get compromised due to internal defects. It causes billions of dollars to the software industry for the maintenance of their products [1], [2]. Whenever a technical dispute occurs in a software environment, the problem is detected, as a bug by a Bug Tracking System (BTS), for instance, Bugzilla[1], and a bug report is generated. Hence, openly accessible software, such as Mozilla, Eclipse, Apache, and Open Office creates a substantial amount of duplicate bug reports. The BTS prioritizes every bug report by its severity and assigns them to the software developers for further assistance accordingly. The major issue in the process is, chance of multiple bug report generation that may have similar kinds of concern, resulting in an immense probability to possess an identical or correlative solution over and over. This emerges the problem of duplicate bug reports [3], [4]. These bug reports are written in natural language in the form of texts, which is usually ambiguous that makes it harder to hunt down the duplicates, as different vocabulary can be used to describe the same defect. Due to this obstacle, the developer ends up striving repetitively for obtaining the same solutions for different bug reports, whereby software companies end up spending over 45% of the cost for fixing these bugs [1]. On that note, if more than one bug has the same solution, then the first report is being marked as a master report and the rest would be marked as duplicate reports of that particular master report. If there is an existing master report of incoming bug report in the repository and still that report is directed to the developer, it will consume lots of resources for an issue that is already or in process of being solved. As a result, detecting the duplicate bug reports is extremely essential.

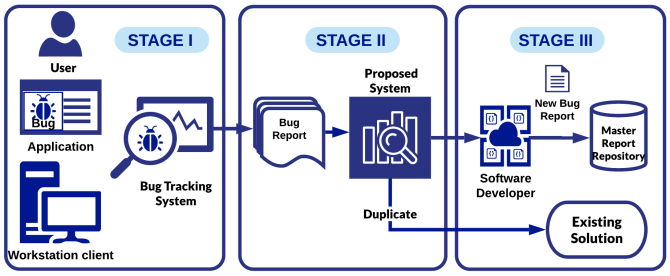


Fig. 1: A Typical Software Bug Handling Flow: Stage I - Bug Report Generation, Stage II - Detection of Duplicate Bug Report, and Stage III - The Issue Addressed by the Developer.

Figure 1 shows a typical flow from the generation of bug reports to the operations carried out at the software developer side. In Stage I, the bugs are tracked and reports are generated by a BTS. In Stage II, the proposed model receives the bug reports and determines if they are duplicates of already reported issues in the master report repository. In Stage III, if the new bug report is a duplicate of any of the master reports, the solution is dispatched to the user; otherwise, it is submitted to the developer to seek out a fix and it will be stored in the master report repository, as a new master report. It aims at limiting the number of bug reports that are being reported to the developer. In 2012, approximately 7,600 bugs/month were reported to Mozilla programmers [1], [2]. Recently, Microsoft claims that in average 30,000 bugs are tracked by their BTS per month [5]. Such severe conditions are obviously hard to handle by a manual process. Thus, an automation can increase productivity and reduce the time taken to seek out these duplicate bug reports [6]. However, the generated bug

T. Akilan is with the Department of Software Engineering, and D. Shah, N. Patel and R. Mehta are with the Department of Computer Science at Lakehead University, Thunder Bay, ON, Canada. (e-mail: {takilan, dshah12, npatel38, rpatel30}@lakeheadu.ca).

[1]https://bugzilla.redhat.com/

reports are written in natural language that poses the following challenges when implementing an automatic system for this problem.

- Ambiguity on lexical, syntactic and semantic levels.
- Word sense disambiguation.
- Memory and computation time constraints.

The proposed system focuses on the aforesaid issues, and it intends to be a time and cost effective solution. Furthermore, it reduces the efforts of the software developers for fixing the defect that has already a solution and it allows them to directly deploy the solution instead of struggling to resolve it from scratch.

II. Literature Review

To release the burden on a triager for the detection of duplicate bug reports, many researchers have put their efforts focusing on identifying the duplicate bug reports automatically through exploiting Machine Learning (ML) and Natural Language Processing (NLP).

For instance, Sandusky *et al.* [7] presents an approach that uses formal and informal relationships between reports to identify the distribution of open-source projects. Hiew [8] builds a semi-automated duplicate detection approach using Term Frequency–Inverse Document Frequency (TF-IDF) weight-based clustering to group similar report around a certain number of centroids. That semi-supervised approach achieves precision and recall rate of 29% and 50%, respectively. Similarly, researchers Jalbert and Weimer [9] attempt the TF-IDF-based weighted word vector representation and Cosine distance between weighted word vectors derived from the documents for sorting the similar bug reports. Hence, Sun *et al.* [10] also consider TF-IDF as vector representation and train a Support Vector Machine (SVM) on TF-IDF to retrieve possible duplicates. However, the model records a poor success rate due to lack of intra class discriminative ability of the model.

Tian *et al.* [11] improve the model presented in [9] by using advanced repository (REP) similarity measure, like in [12], rather than the standard Cosine distance. In this direction, Zou *et al.* [13] come up with a new document similarity measure that integrates LDA with Ngram called LNG. By doing so, their model outperforms all the previously reported techniques using LDA and Ngram models, for example, it achieves 11 - 18% higher recall rate when compared to the REP-based algorithm proposed in [9].

On the other hand, Runeson *et al.* [14] study the impact of various preprocessing approaches, like stopword removal, tokenization, stemming, and vector space representation as well as document similarity metrics, such as Cosine, Dice and Jaccard measures for the application of duplicate bug report detection. Hence, Sureka and Jalote [15] apply character level Ngram as a low-level feature to represent the title and description of the bug reports. In this way, they are able to get a recall rate of 33.92% and 61.94% for Top-50 results for 1100 and 2270 test cases, respectively.

Kim *et al.* [3] implement a bug localization using two-phase predictive model. The authors employ the standard bag-of-words for feature extraction from the collected bug reports. Then, in the first-phase, they train a Naive Bayes algorithm on the extracted features for retrieval of Top-N recommended solutions for the reported bug. Sequentially, the second-phase filter out the less informative reports before the actual prediction. Thus, the two-phase predictive model achieves a 70% of correct predictions. Hence, some of the recent works like Minh [16] employ Ngram, cluster reduction based on historical information of the bug reports, and an SVM vector space model to represent bug reports. The author extracts character- and word- level Ngrams to find lexical similarity between words.

All the above works try out different feature representations and document similarity measures for improving the prediction rate, but neglect the timing overhead. Thus, this work concentrates on speeding up the retrieval process via an efficient double-tier topic modeling and classification strategy.

III. The Proposed Model

The proposed solution is a double-tier model towards a computational time efficient bug handling process. Here, when a new report is reported to the model, it will seek out the Top-N similar master reports preserving a great deal of time of the developer to find and dispatch the appropriate solution. By receiving the Top-N recommendations, the developer can discover the best matching solution from the small N number of master reports rather than the large number (generally $> 30K$) of one-on-one comparisons. The proposed double-tier model consists of two stages: Clustering and Classification. The stages are elaborated in the coming subsections.

A. Stage 1: Clustering

This stage comprehends multi-step functionality, which includes preprocessing of the dataset, LDA-based topic modeling, and clustering.

1) Preprocessing: The bug reports are ambiguously written in regular natural language, as discussed earlier. Consequently, it requires a sufficient amount of preprocessing to clean the data and to transform it to a standard format for further analysis. It consists of five main steps, namely text cleaning, stop word removal, tokenization, lemmatization, and generation of Bag of Words (BoW).

a) Text Cleaning: Cleaning of the bug reports includes the removal of unnecessary parts of the text and the elimination of invalid bug reports. Invalid bug report elimination will remove the bug reports that have empty description field as well as a certain fixed patterns of description, for example, the description may have “Fixed in HEAD” and “Has been marked as read-only”. Here, the first phrase represents that the particular issue was revised in previous versions and the second phrase means that the bug report is with read-only permission. So, such bug reports have nothing to be addressed. However, having them inside the model may impact the similarity measures. Therefore, all such reports must be

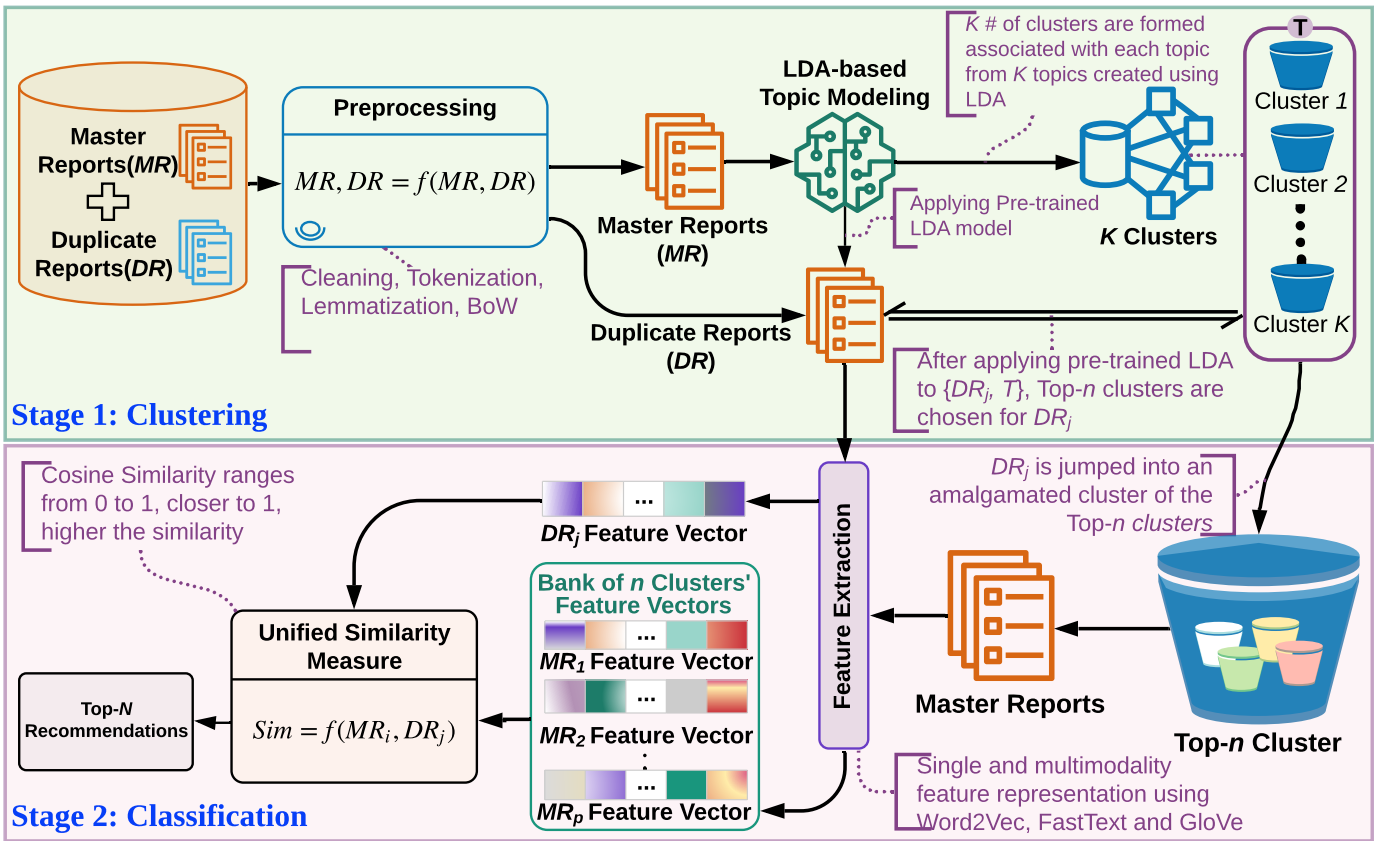


Fig. 2: The Proposed Double-tier Topic Modelling and Classification System for Fast Duplicate Bug Report Detection.

removed before modeling an effective duplicate bug report detection system.

b) Stop Word Removal: There are many terms, viz. "the", "that", "a", "in" etc., which are referred as stop words that do not carry much significance but can adversely affect the duplicate bug detection process. These stopwords have to be removed so that more emphasis can be granted to the key terms with higher significance. After stop word removal, the entire text is converted into lower case.

c) Tokenization: Tokenization is the process of splitting text into minimal considerable elements called "tokens" [17]. All the bug reports are tokenized and the list of tokens for each report will be fed to the subsequent stages as inputs.

d) Lemmatization: Lemmatization derives a word to its root form preserving the context, for example, "fixing" will become "fix", "requires" will become "require".

e) Generation of BoW: BoW is a technique used to extract features from textual data. It is a portrayal of text, which depicts the occurrence of a word within a document. A text is represented, as a bag of its words, ignoring the grammar and even word sequence but allowing multiplicity. There are two steps involved in this process: determination of vocabulary and vectoring the words according to its presence in a document.

2) LDA-based Topic Modeling: LDA is an unsupervised statistical model that allows the class of observations to be

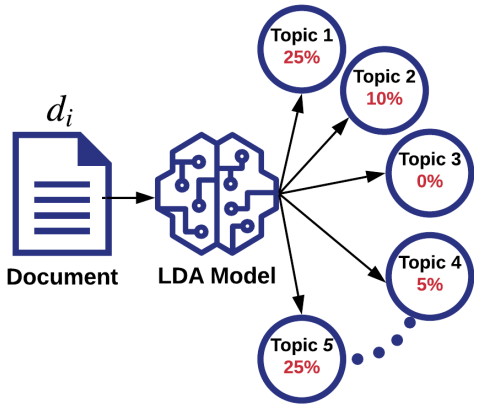


Fig. 3: Example of Topic Modeling with LDA.

interpreted by unseen groups describing why some parts of the data are similar. The underlying concept is that each document is represented as a distribution of latent topics, where each topic is characterized by a distribution of words. By implementing LDA on BOW, a document-topic matrix is generated. Then, for each document, LDA generates a probability distribution of the topic within that document. Figure 3 illustrates an example of probability distribution of topics within a single document. LDA works by first making a key supposition, in which a document is processed for

choosing a set of topics. Then, it assigns set of words to each topic. It subsumes the following six steps.

1) **Initialization:** Number of topics t is set.
2) **Random topic assignment:** Randomly assigning each word v in the document d_i to one of the t topics. This random assignment gives both topic distribution of all the documents and word distributions of all the topics.
3) **Reassignment:** Improving the random assignments by going through each word v in document d_i and determining:
   - The total number of words in document d_i that are presently allocated to topic t.
   - The total number of assignments to topic t over all the documents that come from the word v.

   Finally, the word v is reassigned to a new topic.
4) **Model updation:** LDA supposes that all topic allocations excluding the current word are correct, later, it upgrades the allocation of the current word.
5) **Iteration:** The above steps are iterated until all the assignments are appropriate. By using these assignments, the probability distribution of k topics in document d_i and the words assigned to each topic will be accomplished.

Suppose that, the LDA initializes t topics from a document d_i. Assuming that the prior distribution is a $k \times z$ matrix, and δ is a variable of the distribution of the word (δ is aggregation of words) within the topic, i.e., δ_{ij} = P($x_j|y_i$) is that the probability of the word x_j within the i_{th} topic. Thereby, we create a document topic matrix of N topics expressed, as given by Eq. (1).

$$p(\alpha, y, x) = p(\alpha|\theta) \prod_{n=1}^{N} p(y_n|\alpha)(x_n|y_n, \delta), \tag{1}$$

where x is the vector composed of the N words, y is the topic vector of N dimensions and α is the topic distribution vector of the document. Given that α and y are latent variables which cannot be noticed, they are removed using marginal distribution from the left as:

$$p(y|\theta, \delta) = \int p(\alpha|\theta) \prod_{n=1}^{N} p(y_n|\alpha)(x_n|y_n, \delta) d\alpha, \tag{2}$$

To corpus c, which has m documents $p(c|\theta, \delta)$ is estimated, as

$$p(c|\theta, \delta) = \sum_{c=1...m} p(x_c|\theta, \delta). \tag{3}$$

Thus,

$$p(D|\theta, Z) = \prod_{d=1}^{M} \int p(\alpha_d|\theta)(\prod_{n=1}^{N_d} \sum_{dn} p(\beta_{dn}|\alpha_d) p(\gamma_{dn}|\beta_{dn}, Z)) d\alpha_d. \tag{4}$$

Subsequently, the LDA model maximizes parameters θ and δ in $p(c|\theta, \delta)$ using the following expression.

$$p(\alpha, y|x, \theta, \delta) = \frac{p(\alpha, y, x|\theta, \delta)}{p(y|\theta, \delta)} \tag{5}$$

We obtain p(x|y), θ, and δ after constructing a topic model, and then infer the topic of the new unlabeled text by the trained topic model and anticipate its related topic distribution, which is also the conditional probability distribution of the document in the topic space.

a) Selection of Optimum Number of Topics in LDA: To select the optimum number of topics, coherence score, which is a combination of intrinsic measure UMass and extrinsic measure UCI, is utilized, as given in Eq. (6).

$$coherence = \sum_{i<j} Score(w_i, w_j). \tag{6}$$

The UCI measure uses a Pointwise Mutual Information (PMI), as defined in Eq. (7).

$$score_{UCI}(w_i, w_j) = log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}, \tag{7}$$

where $P(w_i, w_j)$ is the probability of seeing both w_i and w_j co-occurring in a random document, $P(w)$ represents the probability of seeing w_i in a random document. On the other hand, the UMass is an empirical conditional log-probability, as defined in Eq. (8).

$$log(w_i|w_j) = log \frac{P(w_i, w_j)}{P(w_j)}. \tag{8}$$

It can be estimated via document counting, like

$$Score_{UMASS}(w_i, w_j) = log \frac{D(w_i, w_j) + 1}{D(w_i)}, \tag{9}$$

where $D(w_i,w_j)$ is the count of documents containing both words w_i and w_j, D is the total number of documents in the corpus, $D(w_i)$ is the count of documents containing the word. After creation of the topic model, the proposed model generates clusters around the topics.

3) Clustering premised on Topic Modeling: As displayed in the Fig. 2, the dataset is an amalgamation of master and duplicate bug reports. So, this work firstly separates the master bug reports from the duplicate bug reports, as for testing the model on the duplicate reports. Then, the LDA model is trained on the BoW corpus created using the master reports. It outcomes the probability distributions of the topics within each document and each topic in each and every master report. After analyzing the topic distribution in a master report, it will be assigned to the cluster, in which it has the highest topic distribution. The following example describes the cluster formation process.

- Suppose that the LDA generates five topics, as shown in the Fig. 4.
- The topic distribution of Master Report$_i$ (MR_i) is analysed based-on the topics created above.
- MR_i is assigned to the cluster 3, since topic 3 has the highest probability distribution among all the topics.

This process will reiterate on each master report until all the master reports are allocated to their associated clusters.

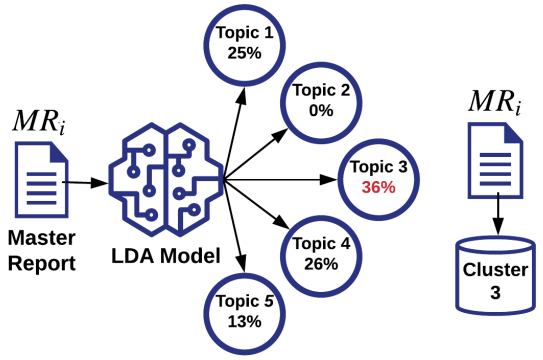


Fig. 4: Example of Topic Modeling Cluster Formation.

B. Stage 2: Classification

This stage comprises multiple steps that involve selection of Top-n clusters, single and multimodal text representation using W2V, FT and GloVe, and text similarity measure fusing Cosine and Euclidean similarity measures.

1) Selection of Top-n Clusters: As shown in Fig. 2, when a duplicate report enters, a pre-trained LDA model is applied to that report. It computes the probability distribution of the topics in each duplicate report. At this point, by adopting the same technique, as used in Section III-A3 for the duplicate report, the topic with the highest probability distribution is identified. But unlike inserting the report into that cluster, the duplicate report is compared to that cluster, as discussed in Section III-B3. If the size of data is huge, there might be a possibility that the associated master report may not exist in the selected cluster. For overcoming this limitation, we select Top-n clusters. Top-n means the n clusters whose associated topics has the first n maximum probability distributions. So, in case, if the corresponding master report fails to exist in the Top-1 cluster, it will also consider next possible cluster to find the appropriate master report. After that, the proposed model uses multimodal approach for forming generalized feature vectors.

2) Formation of Feature Vectors: The formation of feature vector is carried out in a single modality and multimodality fashion using various text to numeric representation algorithms.

Single-Modality Feature Extraction - It employs the following three feature extractors: Word2Vec (M1), FastText (M2), and GloVe (M3).

Multi-Modality Feature Extraction - The multimodal approaches are known for their richness of information extraction and learning [18]–[20]. Thus, the proposed approach exploits multi-modality feature extraction by integrating multiple feature vectors to enhance the performance. Consequently, four multi-modal feature representations are introduced, namely fusion of FastText and GloVe (M4), fusion of GloVe and Word2Vec (M5), fusion of FastText and Word2Vec (M6), and fusion of FastText, GloVe and Word2Vec (M7).

Equation (10) express the fusion of multiple feature vectors.

$$\bar{v} = f_n(v_1^{\mathbf{R}\in D}, v_2^{\mathbf{R}\in D}), \tag{10}$$

where v_1 and v_2 are the vectors formed by different single modality feature extractors, $n \in 1, \cdots, 4$, and D is the dimension of the vectors. The operations employed for amalgamation of the feature vectors can be defined as:

$$f_1(v_1, v_2) = v_1 \oplus v_2, \tag{11}$$

where $\oplus$ is a concatenation operation.

$$f_2(v_1, v_2) = avg(v_1, v_1), \tag{12}$$

whereby the point-wise average of the two feature vectors is acquired. Besides, the Principal Component Analysis (PCA) is also used for dimensionality reduction and enhancing information interpretability with minimal data loss. It does so by creating new unlinked variables that successively maximize the variance. Thus, the PCA is applied on the concatenated resultant from Eq. (11), like in Eq. (13).

$$f_3(v_1, v_2) = PCA(f_1(v_1, v_2)). \tag{13}$$

Similarly, the feature vector generated through Eq. (12) is also reduced with PCA, as given below.

$$f_4(v_1, v_2) = PCA(f_2(v_1, v_2)). \tag{14}$$

3) Top-N Recommendations: After the selection of Top-n clusters, the corresponding master report for the duplicate report is to be retrieved from the Top-n clusters. The core task in this step is finding the accurate similarity measure between the reported bug report and the master report repository. In this case, the unified similarity measure explained in Section III-B3a and Section III-B3b are recalled. The duplicate report is compared to every master report in the Top-n clusters. As a consequence, feature vectors of the duplicate report and each master report from the Top-n clusters are created followed by the average calculation of cosine and euclidean similarities between both feature vectors.

a) Cosine Similarity: It is a similarity measure between two non-zero vectors in an inner product space, as given in Eq. (15).

$$Sim_{cos}(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\|\|d_2\|} = \frac{\sum_{i=1}^{n}(d_{1i}d_{2i})}{\sqrt{\sum_{i=1}^{n}(d_{1i}^2)}\sqrt{\sum_{i=1}^{n}(d_{2i}^2)}}, \tag{15}$$

where d_{1i} and d_{2i} are the vectors of documents d_1 and d_2 in i_{th} dimension topic space. It substantially evaluates the Cosine angle between two vectors that are projected to the multidimensional space. The Cosine value ranges from 0 to 1. The higher the angle, the lower the Cosine value. The Cosine similarity work as follows. Firstly, the given documents are projected into a vector space and generates vectors for each document. Later, the cosine angle between two documents can be measured such a way that the closer the documents by the angle, the higher the Cosine similarity.

TABLE I: Performance Comparison: Proposed Double-tier Topic Modeling and Classification Strategy vs Conventional Approach wrt Recall Rate (RR) and Per Sample Processing Time (PSPT).

		The Proposed Model (w/ Topic Modeling)						Conventional Approach (w/o Topic Modeling)					
Top-N		**10**	**100**	**500**	**1000**	**2000**	**2500**	**10**	**100**	**500**	**1000**	**2000**	**2500**
M1	**RR** (%)	14.5	29.5	44.0	54.0	61.5	62.0	15.5	33.0	48.5	55.0	62.0	65.0
	PSPT (s)	7.356	7.359	7.359	7.401	7.353	7.443	26.274	25.704	25.269	25.515	25.704	25.911
M2	**RR (%)**	16.0	30.5	45.5	55.5	63.0	64.5	16.5	35.0	50.0	57.5	64.0	67.0
	PSPT (s)	7.380	7.360	7.390	7.410	7.420	7.460	25.713	25.944	26.667	26.694	26.850	26.055
M3	**RR** (%)	12.5	23.5	37.5	46.5	55.0	58.0	12.5	27.5	42.0	49.5	59.0	62.0
	PSPT (s)	5.283	5.286	5.322	5.346	5.400	5.409	17.340	17.355	17.451	17.424	17.517	17.436
M4	**RR** (%)	**13.5**	**31.5**	**49.5**	**57.0**	**64.5**	**67.0**	**16.5**	**36.0**	**54.0**	**59.5**	**68.5**	**69.0**
	PSPT (s)	9.257	9.438	9.318	9.3462	9.468	9.380	29.999	30.495	28.162	28.283	28.286	29.975

b) Euclidean Similarity: It is just a straight-line distance from one point to another point in Euclidean space, as given by Eq. (16).

$$Distance_{euc}(d_1, d_2) = \sqrt{\sum_{i=1}^{n}(d_{1i} - d_{2i})^2}, \qquad (16)$$

where d_{1i} and d_{2i} are the vectors of document d_1 and d_2 in Euclidean space $\in \mathbb{R}^n$. From this, the Euclidean similarity score between the two documents d_1 and d_2 is computed, as

$$Sim_{euc}(d_1, d_2) = \frac{1}{1 + Distance_{euc}}. \qquad (17)$$

Finally, the Top-N recommendations are made based on the highest similarity scores.

IV. Experimental Analysis

A. Dataset

The ablation study uses the Eclipse dataset[2]. It is collected from the year 2001 to 2013 consisting of 85,156 bug reports. After all the preprocessing steps performed as elaborated in Section III-A1, the dataset is left with 81,618 bug reports, from which 69,535 are master reports and 12,083 are duplicate bug reports. In this case, all 69,535 master reports and a randomly selected 200 duplicate reports are used. Note that a bug report is a structured document that includes eleven fields, as given below but this study exploits only the description field.

i. **Issue id**: It is the unique number assigned to the bug report by the bug tracking system.
ii. **Priority**: It is the importance and urgency of resolving an error. The higher the priority is, the faster it needs to be solved. It ranges from P0 to P5 with P0 as the highest priority and P5 as the lowest priority.
iii. **Component**: It shows in which part of the system, a problem is raised.
iv. **Duplicated issue**: It shows the duplicate report of the corresponding master report.
v. **Title**: Stating the problem in a single line.
vi. **Description**: This elaborates the problem furthermore.
vii. **Status**: It shows the status of the bug whether it is Open, Fixed, Closed or Deferred.
viii. **Resolution**: It shows the current status of the bug report as if the bug is Fixed, Wontfix, Invalid or Duplicate.
ix. **Version**: It shows the version of the software in which the bug is raised.
x. **Created time**: It shows the time that the bug report is created.
xi. **Resolved time**: It shows the time when the bug was resolved.

[2]https://bugs.eclipse.org/bugs/

B. Computing Environment

The experimental analysis is conducted on a computer having Windows version 10 OS with an Intel Core i7-8550U 1.8 GHz processor and 12 GB of memory.

C. Evaluation Metric

This study is evaluated using the metric defined by Equation (18).

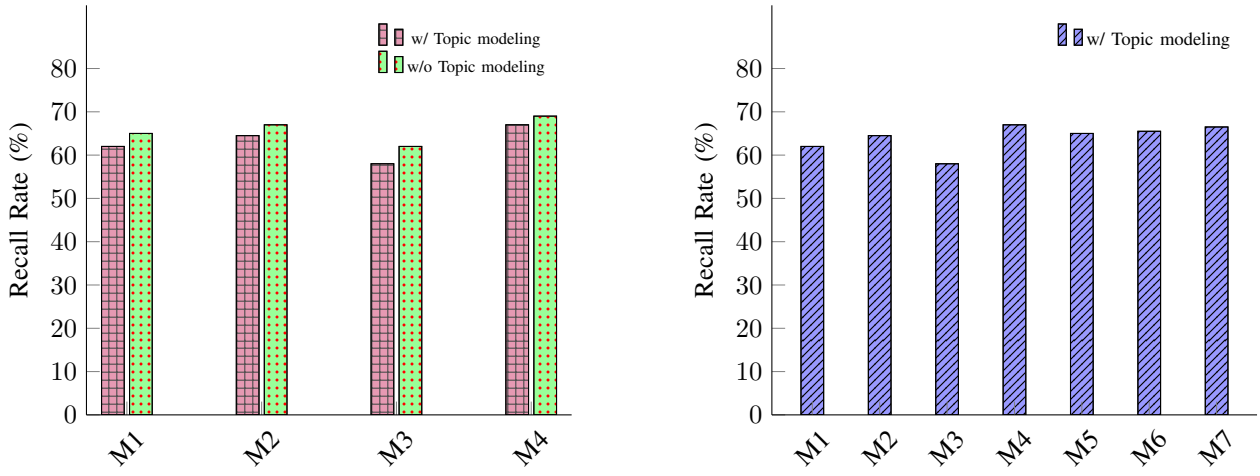
$$RecallRate = \frac{N_{true}}{N_{total}}, \qquad (18)$$

where N_{true} is the number of duplicate reports that can correctly find its associated master report, and N_{total} is the total number of duplicate reports.

D. Step-by-Step Quantitative Analysis

1) Analysis for Finding the Optimal Number of Topics in LDA Modeling: Firstly, the LDA model is trained on the master reports for 20 passes and 100 iterations. After training the model, coherence score is calculated, which is elaborated in Section III-A2a, as a measure to evaluate the optimum number of topics. A sanity analysis was performed to determine the optimum number of topics along with the coherence score. As shown in Table II, for ten topics, we get the highest coherence score as well as the highest recall rate. Consequently, ten is selected, as the optimum number of topics, in this study. Note that in the interest of saving useful computational time, this sanity check prefers the M2 feature extractor over M4 because obtaining RR using M4 is time-intensive in contrast to M2.

2) Impact of Vector Fusion: To find out the best fusion model, experiments are carried out for getting the Top-$2.5K$ recommendations. According to the results tabulated in Table III, the model M4 with the fusion f_4 achieves the highest RR of 67%. Using this as an empirical proof, further



(a) RR of the models

(b) RR across all models

Fig. 5: Performance Comparison of all the Models for Top-2.5K Recommendation.

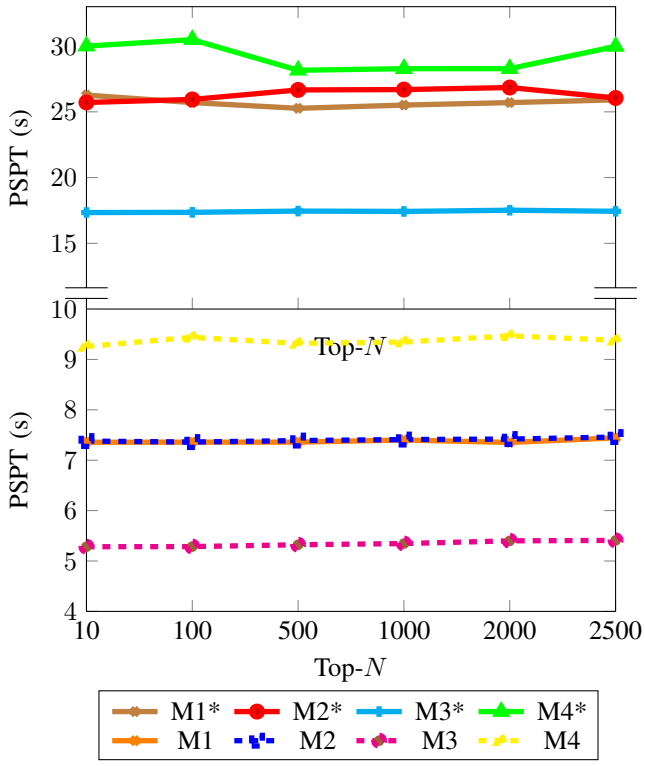


Fig. 6: Per Sample Processing Time of all the Models: (*) - without Proposed Topic Modeling.

experiments are conducted on the models M1, M2, M3 and M4 for finding various Top-N: 10, 100, 500, 1K, 2K, and 2.5K.

3) Performance Analysis: Table I and Fig. 5a compare the efficiency of the proposed double-tier duplicate bug report detection model with the conventional one-on-one classification model. The comparisons are carried out on single modality models: M1, M2, and M3, and the best multimodal technique,

TABLE II: The Results of Sanity Analysis for Finding the Optimal Number of Topics in LDA Modeling.

No. of Topics	Coherence Score	RR using M2 (%)
10	**0.621**	**67.0**
20	0.593	64.5
30	0.582	62.5
40	0.509	63.5
50	0.476	60.0
60	0.472	58.5
70	0.455	57.5
80	0.445	55.5

TABLE III: Performance Analysis wrt to Recall Rate (%) of Multimodal Feature Fusion for Top-2.5K.

Model	f_1: Eqn.(11)	f_2: Eqn.(12)	f_3: Eqn.(13)	f_4: Eqn.(14)
M4	**65.5**	**66.5**	**66.5**	**67.0**
M5	62.0	63.5	64.0	65.0
M6	64.0	62.5	66.0	65.5
M7	64.0	62.5	65.5	66.5

M4 that is chosen from the earlier analysis discussed in Section IV-D2. Here, M1 and M2 utilize CBoW, as the backbone and trained for 100 iterations. On the other hand, M3 produces better results when it is trained for 1K epochs. It is found that among the single modality models, M2 performs better than M1 and M3. For example, the proposed approach using the M2 as feature extractor achieves 64.5% RR for Top-2.5K recommendations. But, when using M1 and M3, the proposed model gets 62% and 58% RR respectively for the same Top-N recommendations. Similarly, the feature extractor M2 is found to perform better than the other two feature extractors in the conventional approach as well.

Figure 5 and 6 analyze the performance of various models in terms of Top-2.5K RR and PSPT, respectively. As discussed and compared earlier wrt Table I and the above Figures, it

is observed that the model M4, with the proposed double-tier strategy, records the better throughput than any other models. In overall analysis, across all the models, the proposed LDA-based topic modeling and classification strategy achieves a competitive performance when compared to the conventional one-on-one similarity-based classification method. Additionally, the proposed approach speeds up the computation by three times. For instance, the proposed model using M4 as feature extractor takes only 9.4s for Top-$2.5K$ recommendations. On the contrary, the conventional technique consumes 30s for the same task.

V. Conclusion

This paper proposes a time-efficient double-tier duplicate bug report detection system using LDA-based topic modeling and classification. Extensive ablation study proves that the proposed model is highly efficient in terms of processing time and the Top-N recall rate than the conventional one-phase approach. However, the recall rate of the proposed approach can be improved in the direction of lower number of Top-N with higher retrieval rate. It is dedicated for future work. The future work also will explore better document similarity measures and document representation techniques.

Acknowledgment

This work acknowledges the organizer of Eclipse dataset and the anonymous reviewers for the technical comments and suggestions.

References

[1] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, pp. 429–489, 2002.

[2] J. Sutherland, "Business objects in corporate information systems," *ACM Computing Surveys (CSUR)*, vol. 27, no. 2, pp. 274–276, 1995.

[3] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where should we fix this bug? a two-phase recommendation model," *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1597–1610, 2013.

[4] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?," *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

[5] T. Warren, *How Microsoft tackles the 30,000 bugs its 47,000 developers generate each month*. PhD thesis, The Verge, 2020.

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

[7] R. J. Sandusky, L. Gasser, and G. Ripoche, "Bug report networks: Varieties, strategies, and impacts in a f/oss development community.," in *MSR*, pp. 80–84, IET, 2004.

[8] L. Hiew, *Assisted detection of duplicate bug reports*. PhD thesis, University of British Columbia, 2006.

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

[10] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 45–54, ACM, 2010.

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

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

[13] J. Zou, L. Xu, M. Yang, M. Yan, D. Yang, and X. Zhang, "Duplication detection for software bug reports based on topic model," in *2016 9th International Conference on Service Science (ICSS)*, pp. 60–65, IEEE, 2016.

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

[15] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *2010 Asia Pacific Software Engineering Conference*, pp. 366–374, IEEE, 2010.

[16] P. N. Minh, "An approach to detecting duplicate bug reports using n-gram features and cluster chrinkage technique," *Int. J. Sci. Res. Publ.(IJSRP)*, vol. 4, no. 5, pp. 89–100, 2014.

[17] B. Patel, H. Balvantrai Patel, M. Khanvilkar, N. Rajendrakumar Patel, and T. Akilan, "ES2ISL: an advancement in speech to sign language translation using 3D avatar animator," in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (IEEE CCECE 2020)*, (London, Canada), Apr. 2020.

[18] T. Akilan, Q. J. Wu, A. Safaei, and W. Jiang, "A late fusion approach for harnessing multi-cnn model high-level features," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 566–571, IEEE, 2017.

[19] T. Akilan, Q. J. Wu, and H. Zhang, "Effect of fusing features from multiple dcnn architectures in image classification," *IET Image Processing*, vol. 12, no. 7, pp. 1102–1110, 2018.

[20] Y. Yang, J. Q. Wu, X. Feng, and A. Thangarajah, "Recomputation of dense layers for the performance improvement of dcnn," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.