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 user applications. Whenever it detects an anomaly situation, it generates a bug report and sends it out 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 at the developer side. Such instances may occur repeatedly in an overwhelming number. This poses a big challenge to the developer. Thus, early detection of duplicate bug reports has become an extremely important task. 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 text similarity measure fusing Cosine and Euclidean measures. 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 only considers 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 in 3 times faster computation than the conventional one-on-one classification model.**

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

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 software developers for the maintenance of their products [1], [2]. Whenever a technical dispute occurs in a software environment, the problem is recorded as a bug by a Bug Tracking System (BTS), for instance, Bugzilla[1], and a bug report is generated. Hence, open-source software, like 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 into 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 text, 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 spend over 45% of the cost for fixing these bugs [1]. 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.

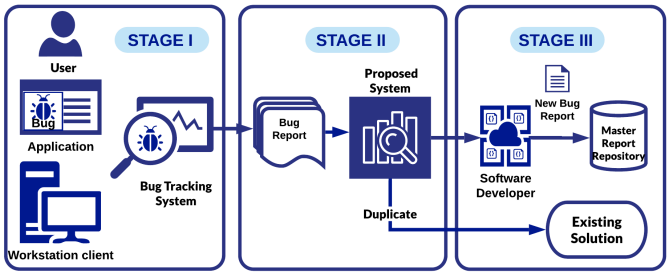


Fig. 1: A Typical Software Bug Handling Flow: Stage I - Bug Report Generation, Stage II - Detection of Duplicate Bug Report, and Stage III - The Issue Addressed by the Developer.

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 the 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 master report, 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 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]. Since, the generated bug reports are written in simple natural language, it poses the following challenges when implementing an automatic system for this problem.

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

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

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

The proposed system is a solution to this matter, which is intended to be time-efficient and cost-effective. Furthermore, it reduces the efforts of the software developers from fixing the defect that has already a solution. Thus, it allows them to just 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 and natural language processing. For instance, Sandusky *et al.* [7] present 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 TF-IDF weight-based clustering to group similar report around certain number of centroids. That semi-supervised approach achieves precision and recall rate of 29% and 50%, respectively. Similarly, researchers Jalbert and Weimer [9] the TF-IDF-based weighted word vector representation and Cosine distance between weighted word vectors derived from the documents for sorting our the similar bug reports. Hence, Sun *et al.* [10] also considers TF-IDF as vector representation and train a Support Vector Machine (SVM) on TF-IDF to retrieve possible duplicates. However, the model reports a poor success rate.

Based on Jalbert and Weimer's [9] experimental setup and approach Tian *et al.* [11] used Mozilla dataset for evaluation and they identified true and false positives, but for similarity measures unlike Jalbert and Weimer [9], Tian *et al.* [11] improves the model introduced in [9] by using advanced repository (REP) similarity measure as in [12], rather than the simple Cosine distance. In the direction of forming a generalized similarity measure, Zou *et al.* [13] implements a new document similarity measuring algorithm 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 model 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] come up with applying 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] implements 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 filters out the less informative reports before the actual prediction. This the two-phase predictive model, achieves a 70% of correct predictions. Hence, some of the recent works, like Minh [16] employ Ngram, Cluster Shrinkage based on historical information of the bug reports, and the SVM vector space model to represent bug reports. There, the author extracts character- and word- level Ngrams to find lexical similarity between words.

All the above works in the literature focus on 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 time effective 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 for the bug. 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 simple natural language as discussed earlier. Consequently, it requires a sufficient amount of preprocessing to clean the data and get it in a standard format for analysis. There are main five steps: 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 one represents that the particular issue was revised in previous versions and the second one means that the bug report is marked as read-only. 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 removed before modeling an effective duplicate bug report detection system.

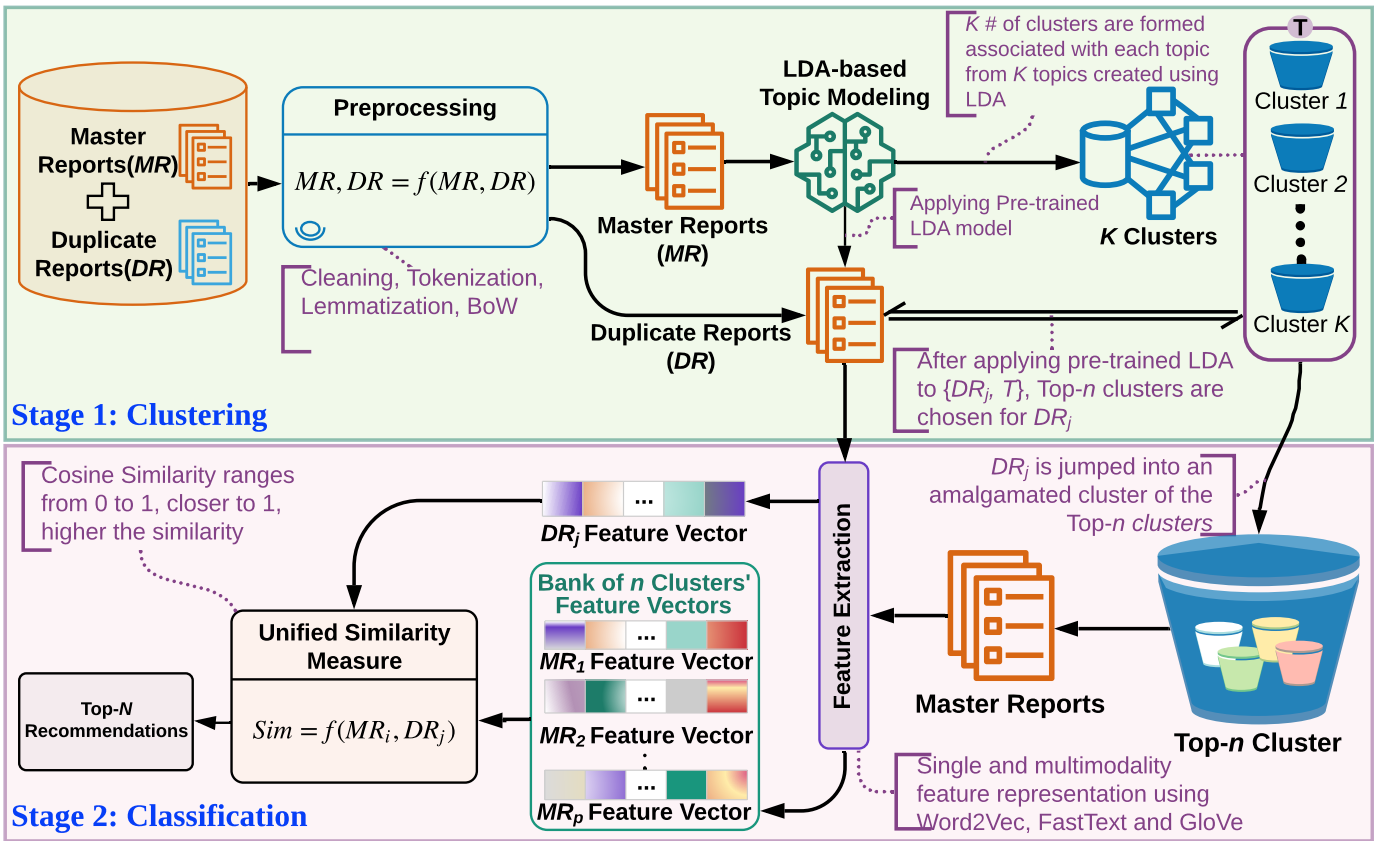


Fig. 2: The Proposed Double-tier Topic Modelling and Classification System for Duplicate Bug Report Detection.

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". 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 keeping 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 interpreted by unseen groups, which explain 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.

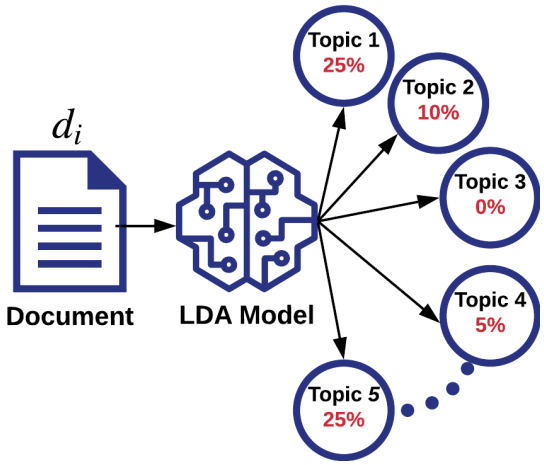


Fig. 3: Example of Topic Modeling with LDA.

LDA works by first making a key supposition on the manner in which a document is processed by choosing a set of topics.

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

1) **Initialization:** Number of topics t is set.
2) **Random topic assignment:** LDA randomly assigns 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:** LDA improve the random assignments by going through each word v in document d_i and determine two things:
   - The amount of words in document d_i that are at presently allocated to topic t.
   - The amount of assignments to topic t over all the documents that come from this word v.

   After the above checking, LDA reassigns the word v to a new topic.
4) **Model update:** LDA assumes that all topic assignments except for the current word are correct, and then it updates the assignment of the current word.
5) **Iteration:** The above steps are iterated until all the assignments seems appropriate. By using these assignments, the probability distribution of k topics in document d_i and the words assigned to each topic can 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 the parameter of the distribution of the word (δ is sum 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. Thus, we generate a document topic distribution of N topics expressed as in 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 α is the topic distribution vector of the document, y is the topic vector of N dimensions, and x is the vector composed of the N words. Since α and y are latent variables which cannot be observed, they are eliminated through 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,

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

Thus,

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

As shown above, the process of constructing the LDA model is to achieve maximized parameters θ and δ of $p(c|\theta, \delta)$.

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

We obtain p(x|y), θ and δ after building a topic model, and then deduce the topic of the new unlabeled text by the trained topic model and predict its corresponding 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 in LDA, Coherence Score is used. Topic coherence given in Eq. (6) is a combination of intrinsic measure UMass and extrinsic measure UCI.

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

The UCI measure uses a score function the Pointwise mutual information (PMI) as defined by 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 uses score function, which is the 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 by simple document counting as given in Eq. (9)

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

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 topic model, the proposed model creates clusters on the basis of the topics.

3) Clustering premised on Topic Modeling: As displayed in the Fig. 2, the dataset is an amalgamation of master bug reports 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 for the quantitative analysis.

Then, the LDA model is trained on the BoW corpus created using the master reports. LDA outcomes the probability distribution of the topics within each document. Hence, LDA provides the probability distribution of each topic in each and every master report. After analyzing the topic distribution in a master report, it will be assigned to the number of cluster in which it has the highest topic distribution. The following example describe the above process of cluster formation in accordance with the topic modeling by an example:

- Suppose, LDA creates five topics. Hence, five empty clusters created as shown in the Fig. 4.

- The topic distribution of Master Report$_i$ (MR_i) is analysed based-on the topics created by LDA.
- MR_i is assigned to the topic that has the heighest probability distribution. In this case, it will be assigned to the topic 3, since the topic 3 has the probability distribution among all the topics.

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

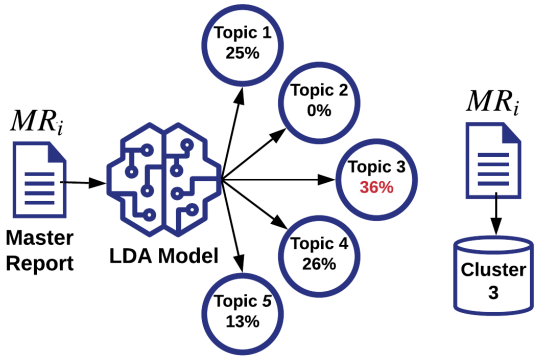


Fig. 4: Cluster Formation premised on Topic Modeling.

B. Stage 2: Classification

This stage comprises multiple steps that involve selection of Top-n clusters, 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 outcomes 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 which will be 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 that limitation, this approach selects Top-n clusters. Top-n means the n clusters whose associated topics has the maximum probability distribution. 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 the process of selection of Top-n clusters, the proposed model uses multimodal approach for forming the 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 individually: Word2Vec (M1), FastText (M2), and GloVe (M3).

Multi-Modality Feature Extraction - 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 as follows: 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 (IV-D2) express the fusion of multiple feature vectors formed by different single modality feature extractors stated earlier.

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

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 are revealed in the following equations:

$$f_1(a, b) = a \oplus b, \tag{11}$$

where b is concatenated with a, which in this case would be v_1 and v_2. Also,

$$f_2(a, b) = avg(a, b), \tag{12}$$

whereby the average of a and b is acquired. Besides, this approach utilizes Principal Component Analysis (PCA) for decreasing the dimension of the feature vector and enhancing the 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 of the Eq. (11) to perform dimensionality reduction as given in Eq. (13).

$$f_3(a, b) = PCA(f_1(a, b)) \tag{13}$$

Similarly, the feature vector generated through the Eq. (12) is also gone through the dimensional reduction as given below:

$$f_4(a, b) = PCA(f_2(a, b)) \tag{14}$$

3) Top-N Recommendations: After performing the process of selection of Top-n clusters as discussed in Section III-B1, 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 used.

The duplicate report is being compared to every master report existing 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 of an inner product space as given in Eq. (15).

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

TABLE I: Performance Comparison of Conventional and Proposed Double-tier Topic Modeling and Classification Strategy 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** (%)	14.5.0	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

where d_{1i} and d_{2i} are the vectors of document 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 bigger the angle, the smaller the cosine value. The cosine similarity work as follows: 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 and it can be analyzed as, the closer the documents by the angle, the higher the cosine similarity.

b) Euclidean Similarity: Euclidean distance is just a straight-line distance between two points in euclidean space as given in Eq. (16).

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

where d_{1i} and d_{2i} are the vectors of document d_1 and d_2 in euclidean n-space. From this the Euclidean similarity measure is derived as:

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

Finally, the Top-N recommendations are made based on the highest similarity scores found from the above computations.

IV. Experimental Analysis

A. Dataset

The Eclipse Dataset[2] is used to analyse the performance of the proposed model. It is collected from the year 2001 to 2013 consisting of 85,156 bug reports. After all the preprocessing steps performed in Section III-A1, the dataset is left with 81,618 bug reports, in which 69,535 are master reports and 12,083 are duplicates of the master 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 various fields (but we use only the description field) as given below:

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 which part of the system a problem is raised.
iv. **Duplicated issue**: It shows the duplicate report of that corresponding master report
v. **Title**: Stating problem in a single line
vi. **Description**: This elaborates on the problem furthermore.
vii. **Status**: It shows the status of the bug-like Open means when it entered, Fixed means when it fixed, Closed means when verified, Deferred means when it postponed and User error means when the user made an error.
viii. **Resolution**: It shows the current status of the bug report as if the bug is fixed, Wontfix, Invalid, Duplicate.
ix. **Version**: It shows the version of the software in which bug is raised.
x. **Created time**: It is the time that the bug report is created
xi. **Resolved time**: It is the time when the bug was resolved

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

B. Experimental Setup

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

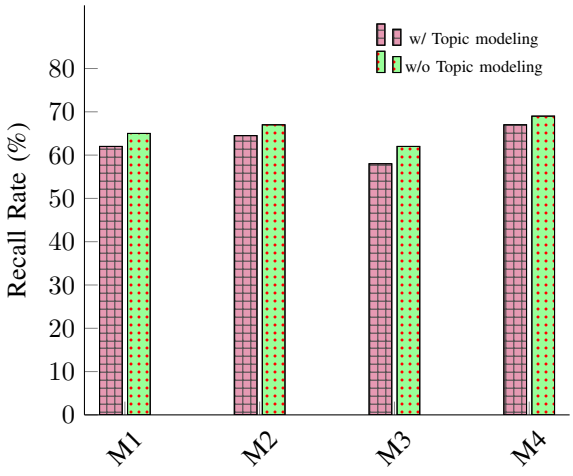
The model is evaluated using the metric defined by Eq.(18), which is the measure of accurately found master reports in the Top-N recommendations for the newly received bug report.

$$RecallRate = \frac{N_{true}}{N_{total}}, \tag{18}$$

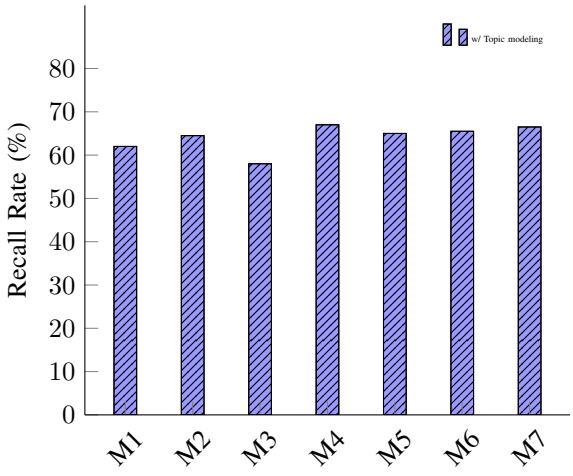
where N_{true} is the number of duplicate reports that can correctly find its associated master report, and N_{total} is the total number of duplicate reports.

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



(a) RR of the models



(b) RR across all models

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

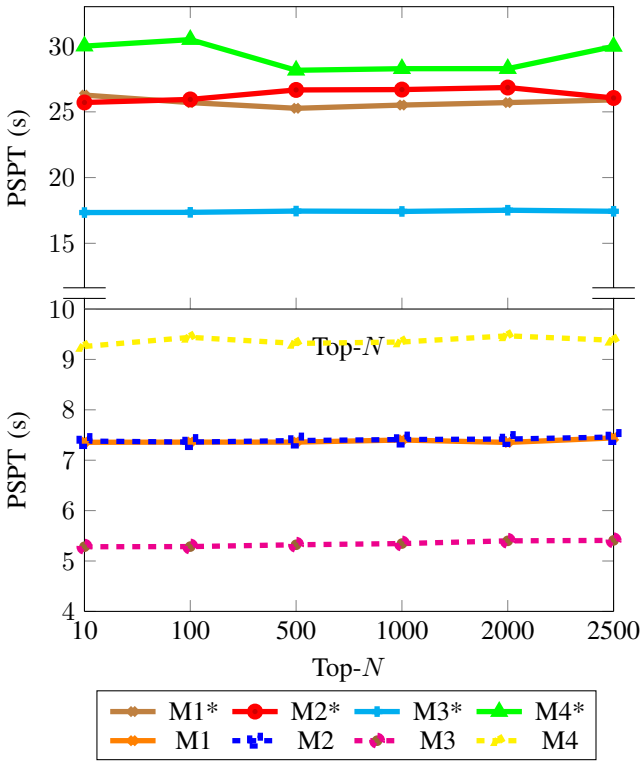


Fig. 6: Per Sample Processing Time of all the Models: (*) - without Proposed Topic Modeling.

the highest coherence score as well as the highest recall rate (RR). Consequently, ten is selected as the optimum number of topics for our dataset. In this analysis, the RR is obtained using the M2 feature extractor.

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

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

3) Performance Analysis: Table I and Fig. 5a compare the efficiency of the proposed double tier duplicate bug report detection approach 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, M4 that is chosen from the earlier analysis discussed in Section IV-D2.

Here, M1 and M2 utilize CBoW as the backbone and are trained for 100 iterations. On the other hand, the M3 produce 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 Figure 6 analyze the performance of various models in terms of Top-$2.5K$ RR and PSPT, respectively. As discussed and compared earlier in association of Table I and the above Figures, it is observed that the model, M4 with the proposed double tier strategy produces 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. However, the proposed approach speeds up the computation by three times. For instance, the proposed model using M4 as feature extractor, it takes only 9.4s for Top-$2.5K$ recommendations out of 69,535 master reports. On the contrary, the conventional technique consumes 30s for the same task. It means the proposed approach records three times higher throughput than the conventional approach.

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 prove 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.

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.