EnBug: Using local environment data and ensemble models in Fault Localization

Justinas Tijunelis
Department of Electrical and Software Engineering
Schulich School of Engineering, University of Calgary
Calgary, Alberta, Canada
justinas.tijunelis@ucalgary.ca

***Abstract*— The only constant in software development is change, which inevitably leads to bugs. Large teams and software systems receive bug report tickets and must find and fix issues as fast as possible. Locating bugs from natural language reports can be a difficult and time-consuming task. When a bug report is submitted, a developer would ideally automate the process of locating source files that are causing the bug. In this paper, we propose EnBug, an information retrieval (IR) and fault localization (FL) method for querying faulty source files with a bug report. EnBug ranks files within a repository based on textual and semantic similarity between bug reports and source code by using an IRFL ensemble of a modified term frequency-inverse document frequency (TF-IDF) and Doc2Vec. EnBug also considers similarities between new and old bug reports to weigh historically faulty source files while ranking. We perform analysis on twelve open-source Java projects with varying numbers of source files and bug reports. The analysis shows that EnBug can effectively locate files with bugs for medium and small projects. This study shows that EnBug can outperform popular techniques and methods such as BugLocator.**

Keywords—fault localization; information retrieval; bug reports; IRFL ensemble

I. INTRODUCTION

Quality management and maintainability are essential software engineering tasks that deliver the promise of shipping high quality and reliable software. Although automated and manual testing, developer tools, and development techniques such as code review have been applied to manage software quality early, software is still shipped with defects. Well managed organizations implement backlogs for bug tracking and reporting that allow developers to understand the replication and details of defects. However, these reports can accrue to thousands of reports and lead to long lead and cycle times in fixing bugs.

Bug reports are submitted and reviewed to a backlog, upon confirmation, a software team will attempt to locate problematic source files and implement fixes. This process is often manual and expensive for large projects, locating bugs requires a strong understanding of a code base. A developer without an understanding code that relates to a bug report will prolong cycle time and decrease their organization's efficiency. The result is expensive, low-value value work that reduces internal and customer satisfaction.

Many researchers have applied information retrieval techniques for querying buggy source files with bug reports [1, 2, 3]. Two common techniques are information retrieval-based fault *localization* (IRFL) and spectrum-based bug fault localization (SBFL). IRFL provides a rank ordered list of source files based on initial bug reports, while SBFL does the same but incorporates program execution information beyond natural language.

In this paper, we propose EnBug, a method inspired by work from Miryeganeh et al. (*GloBug*) that compared performance between various combinations of TF-IDF and Doc2Vec between global and local source file corpuses [4]. From Zhou et al. (*BugLocator*), we utilize a revised Vector Space Model (rVSM) that accounts for document length when ranking files with the assumption that larger files are more likely to contain bugs [5]. We combine rankings from rVSM with a Doc2Vec model to create an ensemble model that can account for term frequency and semantic similarity. Both rVSM and Doc2Vec models also use a historical report similarity weighting proposed by Zhou et al. that presumes new bug reports are related to old reports and thus fixed source files from old reports will be relevant in ranking [5]. We have evaluated EnBug on twelve open-source projects from a subset of *Bench4BL* with varying counts of source files and bug reports [6]. The evaluation showed that EnBug is effective and outperforms *BugLocator*. Our experiments do show that EnBug outperforms its Doc2Vec ensemble model component alone, and that Doc2Vec generally does not perform well in IRFL.

The core ideas behind EnBug are as follows:

1. Leveraging local datasets (i.e., projects) when applying a TF-IDF. The motivation behind this is that developers would want to use FL on a per-project basis. Most developers use multiple languages and paradigms when programming, and requiring configuration to train an FL model with multiple corpuses can hamper the desire to use FL.
2. Capturing semantics using a word embedding-based language model (Doc2Vec) on local datasets. Again,

the motivation for local datasets is provide ease of use to potential users of EnBug. Note that Doc2Vec benefits more from larger datasets, so a tradeoff is made by not training on a global dataset to improve a user's experience.

The main comparison baseline is *BugLocator* [5]. We compare Doc2Vec models and *EnBug* against *BugLocator*. To evaluate these comparisons, we designed and reported an empirical study on twelve open-source projects from *Bench4BL* [6], as follows:

- Heuristic 1: How does incorporating Doc2Vec to *BugLocator* to develop *EnBug* impact performance?
- Heuristic 2: How does Doc2Vec alone perform in an IRFL environment against other techniques like *BugLocator* or *EnBug*?

The contributions of this work are in comparison between *BugLocator* and *EnBug*, replication of *BugLocator's* results with different corpuses, and evaluation of Doc2Vec in IRFL. Additionally, this work covers a method that was not tested by *GloBug*, where TF-IDF and Doc2Vec are used on local datasets (i.e., train both models on individual projects, rather than all projects). This work focuses on using environment based IRFL that would allow developers to apply this work on an individual project without training models with other projects.

In summary, the contributions of this paper are:

- We evaluate EnBug, a bug localization method that creates an ensemble model with Doc2Vec and TF-IDF. We compare this to *BugLocator*, utilizing similar techniques for file-report and report-report similarities to create rankings. We also replicate the results of *BugLocator* using a new dataset from Bench4BL.
- We cover a combination of TF-IDF and Doc2Vec that was not covered in *GloBug*.
- We apply hypothesis testing to evaluate EnBug against *BugLocator*.

The organization of this paper is as follows. In Section II, we cover core FL techniques utilized in EnBug including rVSM and Doc2Vec, general FL techniques, and the design and of EnBug. Additionally, we cover relevant formulae and combination of techniques. In Section III, we cover the methodology of creating and evaluating *EnBug*. In Section IV, we cover experimental design and Section V provides results of experimentation with visualizations and hypothesis testing. Section VI discusses results and compares results between Doc2Vec, *EnBug*, and *BugLocator*, but also discusses results from *GloBug*. We discuss related work in Section VII and discuss potential improvements to *EnBug* from this work. Finally, we conclude our results in Section VIII and discuss potential future directions to arrive at better results.

II. Background

A. IR-Based Fault Localization (IRFL)

IRFL is a technique that aims to produce a ranked set of source files from a bug report acting as a query. Bug reports should contain a detailed description of the fault and outline replication steps. The assumption is that natural language from bug reports can be scored on similarity against source code.

There are several techniques have been applied but Miryeganeh et al. describe the direct and indirect relevancy functions based on the work of *BugLocator* [4, 5]. The direct relevancy is described as the similarity between a bug report query and source files within the reports corpus (local or global). The indirect relevancy is described as the similarity between new and historical bug reports and their respective source file fixes; new bug reports have a similarity score with old reports, and thus an indirect relevance to the old report's file fixes.

The general approach to creating an IRFL solution follows:

- **Corpus Formatting:** First we must retrieve our corpus, on which we format text to remove punctuation, uncommon (stop) words, and other elements that may impact the performance of IRFL. The text is tokenized, and tokens are stemmed. The corpus for bug localization includes project source files.
- **Query Formatting:** Our queries must be formatted in the same manner as the corpus for an appropriate comparison. For bug localization, the queries are bug reports that are composed of natural language and potentially stack traces or other computer language.
- **Data Engineering:** The data for corpuses and queries must be formatted and built into data structures that will enable efficient and simple algorithms for applying functions between data. Documents and queries should be indexed for ease of data retrieval.
- **Retrieval and Ranking:** Retrieval and ranking is performed with textual similarity between queries and documents. Various approaches can be taken to compute similarities including direct and indirect relevancy as described above.

B. BugLocator – Baseline Method

BugLocator utilizes direct and indirect relevancy functions using rVSM and a similarity score calculated as query relevance based on historical queries, respectively [5]. *BugLocator* was validated using four open-source Java projects and found that it significantly outperformed other methods including traditional VSM, LDA, and others [A].

BugLocator's rVSM model accounted for a study which revealed that longer source files are more likely to contain bugs. Since traditional TF-IDF will typically lean towards smaller documents in ranking, rVSM was used to reduce weighting for terms that appear frequently in Java (e.g., public). *BugLocator* uses an indirect relevancy function that creates relevancies between new and historical bug reports using a graph structure to weigh all similar bug reports and source files against the new bug report. The direct and indirect relevancy functions calculate a variably weighted average to obtain a final score.

The *BugLocator* rVSM function is an altered from of the TF-IDF formula (Equation 1) computes the relevance between a bug report (query) and a document using an inverse logit function multiplied by a cosine similarity between query and file TF-IDF vectors.

$$rVSM(q,d) = \frac{1}{1+e^{-N(\#terms)}}$$

$$\times \frac{1}{\sqrt{\sum_{t \in d}((\log f_{tq}+1)\times \log(\frac{\#docs}{n_t}))^2}} \quad (1)$$
$$\times \frac{1}{\sqrt{\sum_{t \in q}((\log f_{td}+1)\times \log(\frac{\#docs}{n_t}))^2}}$$
$$\times \sum_{t \in q \cap d}(\log f_{tq}+1) \times (\log f_{td}+1) \times \log(\frac{\#docs}{n_t})^2$$

, where *N(#terms)* refers to the normalized number of terms in the document, f_{tq} refers to the number of occurrences of a term *t* in a query *q*, f_{td} refers to the number of occurrences of a term t in a document *d*, n_t refers to the number of documents that contain the term *t*, and *#docs* refers to the total number of documents in the corpus.

The comparison between natural language in bug reports and source code can lead to an unreliable comparison. As a second step, *BugLocator* computes an indirect relevance with report-report TF-IDF similarities. In cases where a bug report does not have many similar files, the indirect relevancy function can help in ranking files based on similar reports (Equation 2).

The indirect relevancy (IDR) is defined as:

$$IDR(B) = \sum_{All\ S_i\ that\ connect\ to\ F_j} \left(\frac{Similarity(B,\ S_i)}{n_i}\right) \quad (2)$$

, where *B* refers to the query bug report we are finding indirect relevancies for, S_i refers to an element of *S* that is the set of similar bug reports to *B*, F_j refers to an element of *F* that is the set of all fixed source files connected to each element of *S*.

BugLocator linearly combines rVSM and IDR to find a final score (Equation 3).

$$FinalScore = (1-\alpha) \times N(rVSM) + \alpha \times N(IDR) \quad (3)$$

, where α is a weighting factor in the interval [0, 1], *N(rVSM)* refers to the normalized score of *rVSM* for a given indirect relevancy, and *N(IDR)* refers to the normalized score of *IDR* for a given indirect relevancy.

C. Doc2Vec

Doc2Vec was introduced by Quoc Le et. al [7] following Word2Vec by Mikolov et. al [8]. Word2Vec trains a shallow neural network that captures semantic relationships between words by predicting which words are the most likely to appear together. Doc2Vec is an extension of Word2Vec that represents vectors as documents rather than words. There are various implementations of Doc2Vec but typically involve a combination of Continuous/Distributed bag of words (CBOW/DBOW), Distributed Memory Model of Paragraph Vectors (PV-DM), and Skip-Gram. CBOW takes a set of surrounding words (context) and attempts to predict a middle word [7]. Skip-Gram takes a word and attempts to predict the surrounding words (context). Le et al. note that PV-DM results come close to Doc2Vec as a whole [7].

From Le et al., given a sequence of training words $w_1, w_2, w_3, \ldots, w_T$, the objective of a word vector model is to maximize the average log probability [7].

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t | w_{t-k}, \ldots, w_{t+k}) \quad (4)$$

, where *T* is the number of words. The prediction task is typically done through a multiclass classifier, such as softmax. There, we have:

$$p(w_t | w_{t-k}, \ldots, w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \quad (5)$$

, where y_i is the un-normalized log-probability for each output word *i*, computed as:

$$y = b + Uh(w_{t-k}, \ldots, w_{t+k}; W) \quad (6)$$

, where *U*, *b* are the softmax parameters, *h* is constructed by a concatenation or average of words vectors extracted from W.

Words with similar meaning are mapped to a similar position. While TFIDF would not relate "Paris" and "France" at all, Doc2Vec is better suited to detect their relationship. Doc2Vec can capture these semantics and contextual relationships between words. In the context of IRFL, natural language and code can anecdotally benefit from Doc2Vec as paired words and acronyms like window-view and network-UDP often come up in either bug reports and code, but not both.

D. EnBug

EnBug's proposal is combining ideas from *BugLocator*, *GloBug*, and Doc2Vec. From *BugLocator*, *EnBug* applies direct and indirect similarities to compute a final score using TFIDF as a vectorizer for model *A*. Another model, *B*, is created using Doc2Vec to calculate similarities. The models are then combined to form an ensemble (Equation 7).

$$E = (1-\beta) \times N(A) + \beta \times N(B) \quad (7)$$

, where β is a weighting factor in the interval [0, 1], *N(A)* is the result of similarity scores from Equation (3), and *N(B)* is the result of similarity scores from Equation (3), except Doc2Vec is used of TF-IDF for computing scores.

Opposed to *GloBug*, *EnBug* only uses local corpuses (i.e., project files rather than all files from a set of projects). This decision was made intentionally to develop a solution that does not require users to download a slew of other projects to train and use their system. However, a Doc2Vec model can be prebaked. Additionally, *EnBug* does not combine direct and indirect relevancies separately. It creates two models that follow the methodology of *BugLocator*, one model using Doc2Vec, the other with TF-IDF. *EnBug* aims to capture the benefits of TF-IDF and Doc2Vec with a linear combination ensemble.

III. Methodology

EnBug was developed using Python and various libraries. scikit-learn was used for TF-IDF vectorization, cosine similarity, and other functions. Pandas and NumPy were used to store and manipulate data throughout the analysis. Gensim was used for its Doc2Vec library. Notably, multiprocessing was

used on a per project; a 10-core CPU used in experimentation decreased processing time by an order of magnitude.

A. Corpus and Query Formatting

In *EnBug*, corpuses and queries are cleaned in the same way to match tokens. From the *Bench4BL* dataset:

- **Source Code:** For each project corpus, source file data including file names, dates, unprocessed code, and project name are extracted. The unprocessed code is cleaned using regular expressions to remove punctuation, separate camel case variable names, replace upper case letters with lower case, remove numbers, and remove duplicate white space. Additionally, common words in code such as "copyright", "void", and "public" are removed as these are likely to appear in nearly every piece of code. Finally, all words are tokenized and stemmed.
- **Bug Reports:** For each bug report, data regarding report summaries and descriptions, dates, and fixed files were collected. Summaries and description were combined and processed in a similar manner to source files.

B. Data Engineering

The processed data of bug reports, and source files contained undesirable data. Some bug reports referenced source files that no longer exist in their respective project. These reports were dropped from the dataset. Additionally, some source files and bug reports generated no tokens due to an abundance of stop words; these data were also dropped.

Throughout the experimentation, data for bug reports and source files were kept within two Pandas data frames. Intermediary calculations for rankings and evaluation scores were kept within the same data frames. When training for indirect similarities, bug reports were split into training and testing sets based on their dates.

C. Retreival and Ranking

Ranking was split into two workflows, one for creating a TF-IDF model, and another for the Doc2Vec model. Ranking was done on each local dataset (project), no source code or bug reports overlapped. In calculating indirect relevancy, bug reports were split into testing and training sets using a 30/70 split, respectively. The oldest bug reports were put into the training set. Both used *BugLocator's* implementation strategy. For each model:

- **TF-IDF Model:** First, to find direct relevancy, each bug report generated an rVSM set of ranks from Equation (1) by fitting a TF-IDF vectorizer with the set of source files for the project, then, for each bug report, an rVSM score was calculated. For indirect relevancy, scores were computed using Equation (2). Final scores were calculated using Equation (3), and scores using different α values were calculated.
- **Doc2Vec Model:** This model used the same method as TF-IDF but used Gensim's Doc2Vec model to calculate similarities.

After each model's ranks were calculated, an ensemble model was created used different β with Equation (7).

IV. Experimentation

A. Objectives

The objective of this study is to investigate the effect of our two heuristics:

- Heuristic 1: How does incorporating Doc2Vec to *BugLocator* to develop *EnBug* impact performance?
  - RQ1. Does *EnBug* improve performance over *BugLocator*?
- Heuristic 2: How does Doc2Vec alone perform in an IRFL environment against other techniques like *BugLocator* or *EnBug*?
  - RQ2. Does *BugLocator* perform better Doc2Vec when using both direct and indirect relevancy?
  - RQ3. Does *EnBug* perform better than Doc2Vec when using both direct and indirect relevancy?

The above research questions are designed to provide insight into Doc2Vec's overall effectiveness in IRFL and its impact in ensemble IRFL models.

B. Measurements

To analyze the effective of each model, Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP) were used. In answering each research question, Wilcoxon Signed Rank tests [9] were used to investigate any statistically significant differences between models.

MRR is a metric that effectively measures how close desired query results were to the top of resulting rank scores. It is calculated as follows:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{8}$$

, where Q is a set of queries (bug reports). However, since our query can be matched to multiple faulty source files, we calculate an adjusted MRR score. Suppose your expected results are A, B and the resulting ranks for the corpus are A, B, C. Our expected MRR is 1 but using traditional MRR will result in a score of ¾. We fix this by ordering the fixed files for a query in the same order as its respective rank results, then once a query file is hit, we remove the rank at that index. In practice, with expected results of A, B and resulting ranks A, B, C, when A is found, it is removed from the rank list and following file's rank indices are shifted by one.

MAP is a metric that measures precision of a ranked list. It is calculated as by finding the average precision for each query and taking the mean. Average precision is calculated as follows:

$$avgP_i = \sum_{i=1}^{M}\frac{p(i)\times pos(i)}{number\ of\ positive\ instances} \tag{9}$$

, where M is the number of instances retrieved, $pos(i)$ indicates whether the instance in the rank j is relevant, $p(i)$ is the precision at the given cut-off rank j, and is defined below:

$$p(j) = \frac{number\ of\ positive\ instances\ in\ top\ j\ positions}{j} \tag{10}$$

C. Results

In developing the *BugLocator* model, multiple α values were used to develop a plot to view the relationship between α and adjusted MRR and MAP scores:

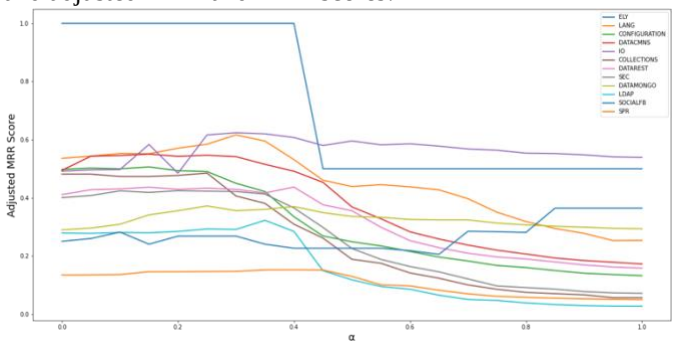

Fig. 1. Adjusted MRR Scores with α=[0, 1] using *BugLocator* model

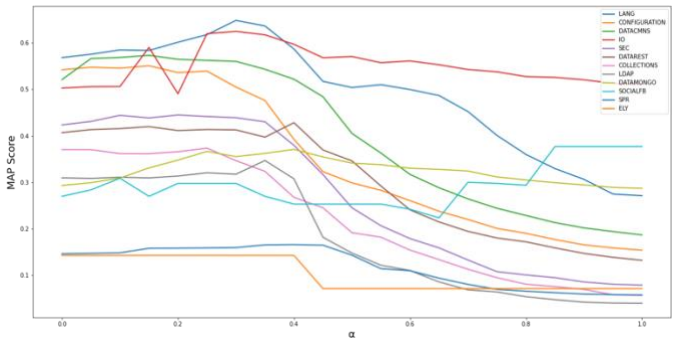

Fig. 2. MAP Scores with α=[0, 1] using *BugLocator* model

As seen in *BugLocator*, α values in range [0.2, 0.3] tend to perform best for both metrics with a falloff as α increases. However, some projects performed better using indirect relevancy only. The "ELY" project was an outlier that had most bug reports refer to files that no longer exist and were thus removed from the dataset.

In developning the Doc2Vec model using the *BugLocator* methodology, multiple α values were used to develop a plot to view the relationship between α and adjusted MRR and MAP scores:

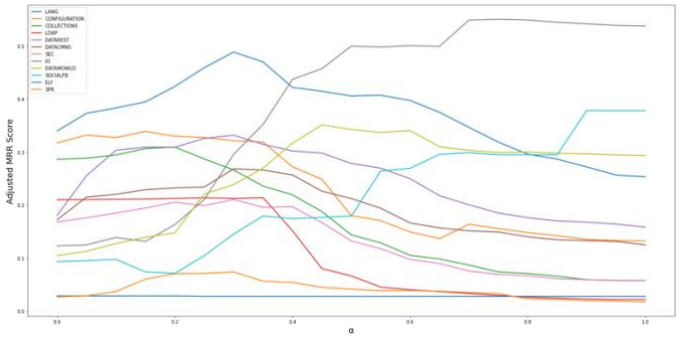

Fig. 3. Adjusted MRR Scores with α=[0, 1] using Doc2Vec model

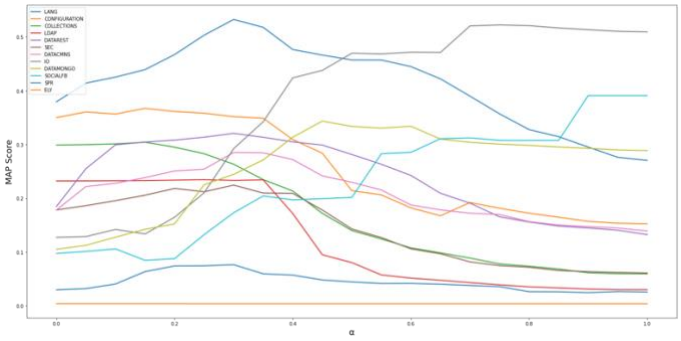

Fig. 4. MAP Scores with α=[0, 1] using Doc2Vec model

The Doc2Vec model follows a similar trend to *BugLocator*, but there is less consistency. Unlike *BugLocator*, α values in range [0.3, 0.4] tend to perform best for both metrics with a falloff as α increases. However, more Doc2Vec projects performed better when using indirect relevance alone in MAP and adjusted MRR.

The scores between the Doc2Vec (Blue), *BugLocator* (Orange), and *EnBug* (Green), are compared below using an α value of 0.3, and β of 0.8 for MAP and Adjusted MRR. Additionally, bug report and source file counts were plotted. :

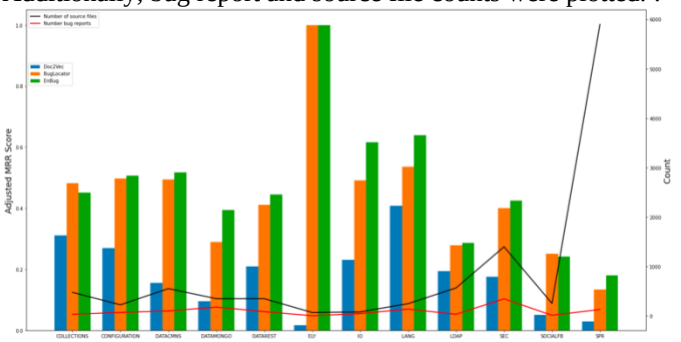

Fig. 5. Adjusted MRR scores per project between Doc2Vec, *BugLocator*, and *EnBug*

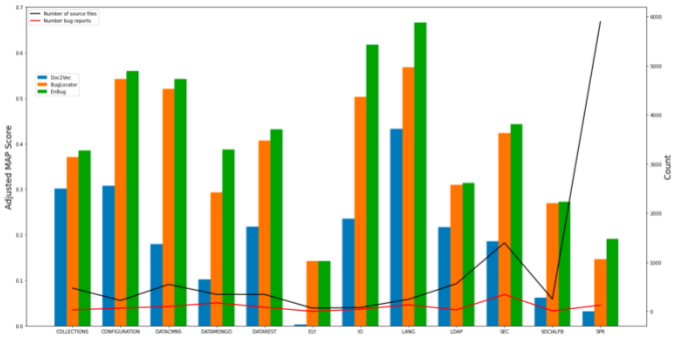

Fig. 6. MAP scores per project between Doc2Vec, *BugLocator*, and *EnBug*

In evaluating RQ1, a Wilcoxon Signed Rank test was performed comparing *BugLocator* and *EnBug* with H_0="The median results of *EnBug* are less than or equal those of *BugLocator*". On adjusted MRR scores, the p-value was 0.03, for MAP scores, the p-value was 0.02, therefore, we reject the null hypothesis and conclude the results *EnBug* outperforms *BugLocator* for both MAP and MRR.

Evaluating RQ2, the same test was performed comparing *BugLocator* and Doc2Vec with H_0="The median results of *BugLocator* are less than or equal to those of Doc2Vec". On adjusted MRR scores, the p-value was 0.0, on MAP scores, the p-value was 0.0, therefore, we reject the null hypothesis and conclude that *BugLocator* outperforms Doc2Vec in both MAP and adjusted MRR.

Evaluating RQ3, the same test was performed comparing *EnBug* and Doc2Vec with H_0="The median results of *EnBug* are less than or equal to those of Doc2Vec". On adjusted MRR scores, the p-value was 0.0004, on MAP scores, the p-value was 0.0002, therefore, we reject the null hypothesis and conclude that *EnBug* outperforms Doc2Vec in both MAP and adjusted MRR.

V. Discussion of results

The study was formed around two heuristics to analyze the effect of using Doc2Vec in IRFL standalone and with an ensemble.

As seen in the results, Doc2Vec alone is not effective in in IRFL, being significantly outperformed by *EnBug* and *BugLocator*. However, in combination with *BugLocator* (creating *EnBug*), we see that the outperformance of *BugLocator* by *EnBug* is statistically significant. By using a β of 0.8, the ensemble of Doc2Vec and *BugLocator* yields superior results, albeit small. The *GloBug* paper notes that Doc2Vec benefits from larger corpuses, but using a linear combination as described in Equation (7) may yield improved results for *GloBug's* most performant combination (Global TF-IDF, Global Doc2Vec).

These results may lead to a re-evaluation with *GloBug*, but some techniques discussed in related work are outperforming it (although, some reports are questionable); effort in other areas may generate more value given the minimal performance gains. These surprising results may provide some insight into using more ensembles in IRFL and even a combination of local and global corpuses within the same model. The complexity of natural language and especially code is ever growing. Utilizing multiple techniques may help IRFL handle more edge cases and improve performance incrementally.

VI. Related work

As discussed throughout the paper, *BugLocator* and *GloBug* are the foundation for the work in this paper [4, 5]. This study built upon *BugLocator* and covered combinations of Doc2Vec and rVSM with local corpuses that were not covered in *GloBug*. *GloBug* found that using global corpuses with Doc2Vec can and rVSM can yield superior results to *BugLocator*.

Other IRFL techniques such as LDA (Latent Dirichlet Allocation) have been applied by Lukins et al., but results are generally poor compared to *BugLocator* [10]. Prior to *BugLocator*, Rao et al. used SUM and VSM, achieving superior results with SUM in 2011 [11]. Poshyvanyk et al. [12] used LSI but *BugLocator* found that it was the least performant of LDA, VSM, and SUM [5].

Jones and Harrold developed *Tarantula*, a spectrum-based FL technique, and incorporated the use of test coverage and prevalence to outperform several other techniques, most of the time [13]. Wen et al. developed HSFL in 2019, building on SBFL by utilizing data of historically suspicious code and source control as well as non-bug inducing repository commits to filter noise [14].

Xiao et al. recently developed *DeepLoc* in 2019 using two conventional CNNs to better capture semantics between bug reports and source files and outperforms *BugLocator* [15]. However, results for *DeepLoc* have not been replicated. Fang et al. published an article in 2021 using an LSTM which learned to filter "informative" and "uninformative" bug reports. While this work did not necessarily create a new IRFL technique, it found that by filtering bug reports, IRFL techniques such as *BugLocator* could be improved [16]. Sangle et al. developed *DRAST*, a combination of AST (Abstract Syntax Tree), rVSM, Random Forest, and several other ML techniques to produce 90%+ MRR and MAP scores on a set of seven selected projects [17]. However, these results have not been peer reviewed.

VII. Conclusion and future work

Overall, *EnBug* was successful in outperforming *BugLocator* and may be used for local IRFL without the overhead of training on a global corpus. This study has shown that using linear combinations in two model ensembles with *BugLocator* and Doc2Vec can yield superior results. Interestingly, Doc2Vec alone drastically underperforms, but a maximized linear combination with *BugLocator* led to improved performance.

Given that using linear combinations of Doc2Vec and *BugLocator* can yield better performance, reimplementing *GloBug* variations with combinations of global and local corpuses and the methodology in this paper may improve results. Using a global corpus with Doc2Vec and local TF-IDF is of highest interest given the strong performance of local *BugLocator* used in this analysis. The *GloBug* team will be contacted with the results of this paper.

Additionally, during the development of *EnBug*, an attempt was made incorporate inter-source file similarities. The idea was that source files are clustered into packages of logic, especially in object-oriented programming languages like Java. The attempt computed source file similarities and reranked the final scores of *EnBug* by weighting file scores by a super position calculated by a file's relative distance to similar files. A publication by Yang et al. applied a similar idea to image processing and found improved performance in image retrieval [18]. Clustering source files and reranking based on inter-rank similarities may be worth pursuing.

In general, this paper has shown that utilizing multiple models rather than finding "the one" can lead to small, but progressive improvements. Work from Fang et al. suggests that improving the quality of queries can yields higher performance for most IRFL techniques [16]. Future work may combine more models like those discussed in Section VI. Additionally, utilizing AutoML techniques could aid in finding optimal combinations between multiple models.

VIII. REFERENCES

[1] D. Poshyvanyk, M. Gethers, A. Marcus, Concept location using formal concept analysis and information retrieval, ACM Trans. Softw. Eng. Methodol. 48 21 (4) (2013) 23:1–23:34. doi:10.1145/2377656.2377660.

[2] S. K. Lukins, N. A. Kraft, L. H. Etzkorn, Bug localization using latent dirichlet allocation, Information & Software Technology 52 (9) (2010) 972– 990. doi:10.1016/j.infsof.2010.04.002.

[3] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, V. Rajlich, Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval, IEEE Trans. Softw. Eng. 33 (6) (2007) 420–432. doi:10.1109/TSE.2007.1016.

[4] N. Miryeganeh, S. Hashtroudi, and H. Hemmati, "GloBug: Using global data in fault localization," *Journal of Systems and Software*, vol. 177, p. 110961, 2021.

[5] J. Zhou, H. Zhang, D. Lo, Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports, in: 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland, 2012, pp. 14–24. doi:10.1109/ICSE.2012. 6227210.

[6] J. Lee, D. Kim, T. F. Bissyand´e, W. Jung, Y. Le Traon, Bench4bl: Reproducibility study on the performance of ir-based bug localization, in: Proceedings of the 27th ACM SIGSOFT International Symposium on Software 49 Testing and Analysis, ISSTA 2018, ACM, New York, NY, USA, 2018, pp. 61–72. doi:10.1145/3213846.3213856.

[7] Q. V. Le, T. Mikolov, Distributed representations of sentences and documents, in: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, 2014, pp. 1188– 1196.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 26, Curran Associates, Inc., 2013, pp. 3111–3119.

[9] M. Neuh¨auser, Wilcoxon–Mann–Whitney Test, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 1656–1658. doi:10.1007/ 978-3-642-04898-2_615.

[10] S. Lukins, N. Kraft and L. Etzkorn. Bug localization using latent Dirichlet allocation. Information and Software Technology ,Volume 52, Issue 9, p. 972-990, September 2010.

[11] S. Rao and A. Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In Proceeding of the 8th working conference on Mining software repositories(MSR'11), ACM, Waikiki, Honolulu, Hawaii, p.43-52, May 2011.

[12] D. Poshyvanyk, Y.-G. Gueheneuc, A. .Marcus, G. Antoniol, V.Rajlich, Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification, Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC 2006), Athens, Greece, p.137-146, June 2006.

[13] J. A. Jones and M. J. Harrold. Empirical evaluation of the Tarantula automatic fault-localization technique. In Proceedings of the 20thIEEE/ACM International Conference on Automated Software Engineering (ASE 2005), Long Beach, California, p. 273-282, 2005.

[14] M. Wen et al., "Historical Spectrum Based Fault Localization," in IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2348-2368, 1 Nov. 2021, doi: 10.1109/TSE.2019.2948158.

[15] Y. Xiao, J. Keung, K. E. Bennin, Q. Mi, Improving bug localization with word embedding and enhanced convolutional neural networks, Information and Software Technology 105 (2019) 17–29.

[16] Fang, F., Wu, J., Li, Y. *et al.* On the classification of bug reports to improve bug localization. *Soft Comput* **25,** 7307–7323 (2021). https://doi.org/10.1007/s00500-021-05689-2

[17] S. Sangle, S. Muvva, S. Chimalakonda, K. Ponnalagu, and V. G. Venkoparao, *DRAST -- A Deep Learning and AST Based Approach for Bug Localization*, Nov. 2020.

[18] F. Yang, Z. Jiang, and L. S. Davis, "Submodular reranking with multiple feature modalities for Image retrieval," *Computer Vision – ACCV 2014*, pp. 19–34, 2015.