2015 30th IEEE/ACM International Conference on Automated Software Engineering

Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports

An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen
Iowa State University, USA
Email: {anlam,anhnt,hoan,tien}@iastate.edu

***Abstract*—Bug localization refers to the automated process of locating the potential buggy files for a given bug report. To help developers focus their attention to those files is crucial. Several existing automated approaches for bug localization from a bug report face a key challenge, called *lexical mismatch*, in which the terms used in bug reports to describe a bug are different from the terms and code tokens used in source files. This paper presents a novel approach that uses *deep neural network* (DNN) in combination with *rVSM, an information retrieval (IR) technique*. rVSM collects the feature on the textual similarity between bug reports and source files. DNN is used to learn to relate the terms in bug reports to potentially different code tokens and terms in source files and documentation if they appear frequently enough in the pairs of reports and buggy files. Our empirical evaluation on real-world projects shows that DNN and IR complement well to each other to achieve higher bug localization accuracy than individual models. Importantly, our new model, HyLoc, with a combination of the features built from DNN, rVSM, and project's bug-fixing history, achieves higher accuracy than the state-of-the-art IR and machine learning techniques. In half of the cases, it is correct with just a single suggested file. Two out of three cases, a correct buggy file is in the list of three suggested files.**

I. Introduction

In a project, software developers spend a great deal of time and effort in debugging and fixing software defects. Those software defects (also called *bugs*) are reported by different stakeholders such as the end-users, testers, or even developers. The stakeholders often report the defects in the documents called *bug reports*. They describe the scenarios in which the software does not perform as expected and provide relevant information regarding the reported defects. After being assigned to repair the defect(s) reported in a bug report, a developer will investigate and locate the source files that are relevant to the defects and potentially defective (also called *buggy*). This process is often referred to as *bug localization* [10].

The process of localizing the defective/buggy files after analyzing the reported bugs is important for developers to quickly and efficiently fix them. However, this process could cost much time and effort. For a large project, developers might have to investigate a large number of source files, make the logic connection from the description of the bug(s) in the bug report to the relevant source files, and leverage their domain knowledge to locate the buggy files. The manual process in order to find and understand the cause of a bug can be time consuming. Thus, it is desirable to have automated tools for bug localization that help developers focus their attention to the potential buggy files relevant to a given bug report.

Several automated approaches have been proposed to help developers in bug localization process. The approaches can be broadly divided into two categories that complement to each other in localizing the bugs. The first line of approaches is fault localization based on the statistics of **program analysis** information [4]. Those approaches rely on analyzing the semantics of the program and/or its execution information with test cases such as passing/failing execution traces. In contrast, the second line of approaches analyzes the content of a given bug report. They use **information retrieval** (IR) [6], [10], [7], [9] or **machine learning** (ML) [5], [8] to automatically search for the relevant, potential buggy files by extracting important features from the given bug report and source files.

The IR-based bug localization approaches share the same principle. The given bug report is construed as a query. The source files in the project are viewed as a collection of documents. The bug localization problem from a bug report is modeled by the searching/ranking problem in IR. Latent Dirichlet Allocation [6], [2] was used to extract topic features for comparison. Zhou *et al.* [10] develop an advanced Vector Space Model, which considers textual similarity between bug reports and files as well as the information on the similar bugs that were previously fixed. A few bug localization approaches from bug reports have been based on **machine learning**. BugScout [8] is a special topic model that assumes that the technical topic(s) described in the given bug report are also those of the buggy files. The topic model is trained to link the topics of the reports and those of source files. Kim *et al.* [5] apply Naive Bayes using previously fixed files as classification labels and use the trained model to assign source files to each report. Ye *et al.* [7] use adaptive learning to rank via features from source files, API description, bug-fixing and change history.

Several researchers [8], [7] have identified *lexical mismatch* between natural language texts in bug reports and technical terms in source code as the key limitation of those aforementioned IR-based bug localization methods. Those ML-based approaches also face the lexical mismatch problem. To improve accuracy, Ye *et al.* [7] and Kim *et al.* [5] use the additional features from the metadata of the reports (e.g., version, platform, priority, etc), and the bug-fixing histories on source files. To bridge the lexical gap, Ye *et al.* [7] additionally use *the texts from the documentation* of the APIs used in the source files. However, API documentation contains texts regarding more general tasks than project-specific buggy behaviors. Kim *et al.* [5] do not use the source files's contents to extract features. They use only the names of the fixed files as classification labels on the reports. Thus, they cannot suggest the files that have *not* been fixed before for a new bug report. BugScout [8], an ML approach, uses only one level of abstraction in topics, thus, might not sufficiently link the terms in two spaces of the

978-1-5090-0025-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ASE.2015.73

bug reports and the source files. In brief, *lexical mismatch is still an issue for existing IR and ML approaches*.

In this work, we build *HyLoc*, a model combining *revised Vector Space Model* (rVSM) [10], an advanced IR technique, with Deep Neural Network (DNN) to recommend the potentially buggy files for a bug report. rVSM extracts the feature to measure *textual similarity* between bug reports and source files. DNN is used to measure the *relevancy* between them by learning to relate the terms in bug reports and potentially different code tokens and terms in source files if they appear frequently enough in the pairs of reports and corresponding buggy files. DNN has been successfully used in other areas, e.g., pattern recognitions, natural language processing (NLP), image and text processing with its capability to capture high-level discriminative information on the input features. We aim to empirically investigate that capability on code features and whether it could lead to bridge that lexical gap.

To extract feature for the relevancy between a bug report and a source file, we use DNN as follows. An DNN contains two separate spaces to handle the features of different nature in the bug reports (text tokens) and in the source files (e.g., identifiers, APIs, comments). After being projected into the spaces with a smaller number of dimensions, the projected features are fed into two DNNs: (1) an DNN to learn the relations between bug reports' texts and the textual tokens in code (i.e., comments); and (2) another DNN to learn between bug reports' texts and code tokens (i.e., identifiers, APIs). We aim to investigate if DNN can emphasize the relations of the text features in bug reports and the related features in source files via its weights.

Generally, a key limitation of DNN is scalability. In this problem, the numbers of examples and features are in the range of hundreds of thousands. To address that, we adapted the DNN-based autoencoder [3] to reduce the dimensions of the feature spaces while maintaining important information and removing redundant one. Each counting feature in each space in the input is projected into a continuous-valued feature space with a smaller number of dimensions. In addition to gaining scalability, we expect that the terms that are "related" in some abstraction will be mapped to nearby locations (at least along some dimensions) in the projected space as shown in NLP [1].

In addition to the *textual similarity feature* computed from rVSM and the *relevancy feature* from the above DNNs, we also integrate the third types of features, called *metadata features*, extracted from the bug-fixing history as in the prior work by Ye *et al.* [7] and Kim *et al.* [5]. For example, a recently fixed file is more likely to still contain bugs than a file that was last fixed long time ago. Thus, we compute the bug-fixing recency score for a file and use it as a metadata feature. To combine those three types of features (textual similarity, relevancy, and metadata), we use another DNN as a *non-linear* feature combinator to compute the final score of a file with respect to a bug report.

Our empirical experiments on several projects showed that combining rVSM and DNN achieves higher accuracy than individual models. HyLoc with the combination of three above types of features improves 2.7–20.7% top-1 accuracy (9.5–102.4% relatively) over the state-of-the-art approach in Ye *et al.* [7] (which has been shown to outperform IR-based BugLocator [10] and the topic-based BugScout [8]). It achieves 16.2-46.4% higher top-1 accuracy than the Naive Bayes approach in Kim *et al.* [5]. We also showed that DNN and rVSM complement well and DNN can link terms in bug reports and different terms and code tokens in relevant source files. This paper contributes:

1. HyLoc, a model for bug localization combining IR and DNN. We also introduce a dimension reduction technique and a feature combination technique using DNN, and

2. Empirical studies on real-world projects to show that the improvement of HyLoc over the existing approaches.

II. A Combining Model for Bug Localization

A. Key Design Ideas

We propose HyLoc, a combination model, with the key ideas:

1. Using DNN to bridge the lexical gap in which features of different types in bug reports and source code are handled in different spaces. Inspired by the success of DNN in capturing high-level abstractions, we expect DNN to bridge the lexical gap by learning to relate the terms in bug reports to the code tokens or comments in source code that might *not* be textually similar to one another. To compute **the relevancy** between a bug report and a file, we treat the reports as texts and extract the textual tokens. In the source code space, we extract the following features: 1) identifiers, 2) APIs (API method calls and classes), 3) textual comments, and 4) textual descriptions of APIs used in the code. Instead of putting them in the same feature space as in existing IR and ML approaches, in our DNN, we collect the features of different types into *separate spaces* and build the feature vectors. Moreover, we use two DNNs: one to learn the relations between bug reports' texts and the textual tokens from source code, and another DNN to learn between bug reports' texts and the code tokens. The rationale for two spaces is that we expect that the DNNs can recognize such relations via the weights among the features of different nature in two spaces if they occurred frequently enough in the pairs of bug reports and the corresponding buggy files.

2. Using another DNN for feature combination. We aim to combine DNN (to handle lexical mismatch) and rVSM [10] (an IR model to measure textual similarity) to complement each other. Moreover, as in Ye *et al.* [7], we also consider the features extracted from the bug-fixing history of a project including the recency and frequencies of bug-fixing files, etc. Instead of using a hill-climbing algorithm for adaptive ranking as in Ye *et al.* [7], we chose to combine the features via another DNN. We expect that with sufficient training data, the weights in the layers reflect the weights of features in the combination. The combination via DNN with non-linear function is expected to perform better than the linear combination in IR-based adaptive learning. DNN will combine 3 types of features (textual similarity, relevancy, and project's metadata).

3. Using another DNN to perform dimension reduction for feature vectors. The number of features extracted from bug reports and source code files during training can be very large. Thus, to scale to real-world projects, inspired by *autoencoder* [3], we built a DNN model to reduce the number of dimensions of the feature space while maintaining important information. The idea is to train the DNN model for projection in which it can encode the original feature vectors in a way that maximizes the similarity between those original vectors and the decoded vectors from the encoded ones.

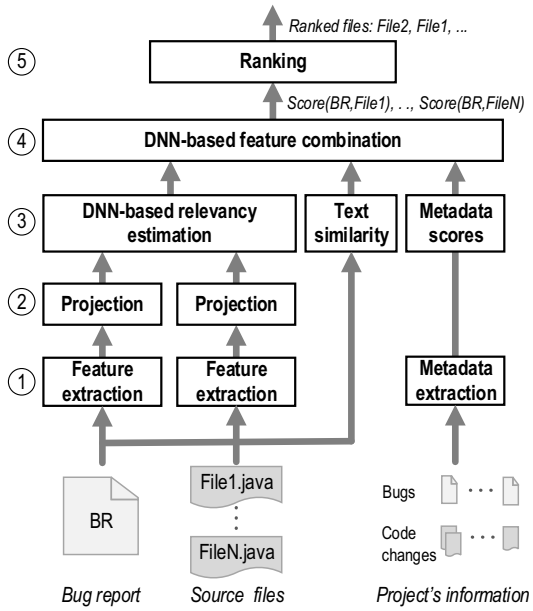

Fig. 1: HyLoc Bug Localization Model

B. Model Architecture

Figure 1 shows the architecture of the entire model. For training, we create a positive pair of a bug report and one of its corresponding buggy files. The pairs are created for all corresponding buggy files and for all bug reports as well. We also create the negative pairs by selecting for every bug report the non-buggy files that are textually similar to the report.

For each pair of a bug report and a source file, we extract the features from them, and build the feature vectors. Each vector has its elements being the significance values of features in the feature types. The feature vectors for each type of features are projected into a continuous space with smaller dimensions using the corresponding autoencoder DNN. All vectors after projection are fed into a DNN to learn the relevancy of each file with respect to a bug report. The output of the DNN relevancy estimator, called *relevancy score*, is fed into the DNN-based feature combinator, which learns the weights to combine three types of features (relevancy computed from DNN, textual similarity computed via rVSM, and the metadata of the bug-fixing history). The weights of the feature combinator help to set the importance of the features in determining the buggy files for a bug report. Note that all the DNNs for relevancy estimation and feature combination in our stack-based architecture can be trained independently one layer at a time where each layer is treated as an unsupervised restricted Boltzmann machine. We take advantage of this key advance of multi-layer architecture in DNN over regular ANN in order to scale HyLoc to large projects. (In an ANN, all the weights must be trained at the same time in the same model).

For prediction, for a given bug report B, we will pair it with each source file f. The features are extracted and feature vectors are built for each pair. Our model produces the final score for f with respect to B. The higher score implies that the model estimates that f is more potentially buggy for B. All the scores for all the source files are used to rank them.

III. DNN-based Relevancy Estimation

Figure 2 shows the DNNs for feature reduction and relevancy estimation between a bug report and a source file. Their feature vectors are fed into the projection DNNs. The outputs of those projection DNNs are concatenated into larger vectors and used as the inputs of the DNN-based relevancy estimator.

The estimator takes those inputs, and outputs a relevant score for a bug report and a file. In the relevancy estimator, we use two DNNs to learn the relations between the features in the bug report and those in the source files. More specifically, the first DNN is used to learn the relations for the text features from bug reports and the source code features such as identifiers and API elements (the names of API classes and methods). We call this DNN Bug Report-to-Code DNN. The second DNN, called Bug Report-to-Text DNN, is to learn the relations between the textual features in the bug reports and those in comments and API descriptions. The reason for the use of two DNNs with the separation between the first group of identifiers and APIs and the second group of comments and API descriptions is that the former is source code while the latter is natural-language texts. The bug reports might contain both and should be matched with both groups. The use of two DNNs Bug Report-to-Code DNN and Bug Report-to-Text DNN is the key difference of our model from the existing approaches, because we want to use DNN to learn the linking of the features with different nature. For each DNN of the relevancy estimator, we use the same DNN architecture as the RBM machine except that we use only one hidden layer. The formula to compute the output of the relevancy estimator is the same.

We expect that the two DNNs for the relevancy estimator would relate the features in bug reports and those in source code, and the features in bug reports and those in comments and API descriptions. The relations are expected to be expressed via the weights from the input features to the nodes in the output of a DNN. For example, "resource", "connection", "session", in a bug report are expected to be strongly related to some hidden nodes that are also impacted much by the nodes representing getCounterRec, ctx, getPermission in the identifier group and by the nodes representing addLocalEjb and sessionExpired in the API group. The outputs of the two DNNs of the estimator are fed into the DNN-based relevancy output module. In training, its output is a score of 0 or 1 where 1 indicates relevancy. In predicting, the relevancy score can have any value between 0 and 1. The higher the score, the higher the confidence the model estimates the relevancy between a bug report and a file.

IV. Empirical Evaluation

We conducted several experiments aiming to 1) evaluate the *accuracy* and time efficiency of HyLoc in localizing buggy files from bug reports; 2) study the impacts on accuracy of HyLoc's *parameters/components* and features; and 3) compare HyLoc to the existing IR and ML approaches for bug localization.

A. Experimental Setting and Metrics

Benchmark DataSet. For comparison, we used the same dataset provided by Ye *et al.* [7] (Table I). All the bug reports, source code links and buggy files, API documentation, and the oracle of bug-to-file mappings are all publicly available at http://dx.doi.org/10.6084/m9.figshare.951967 thanks to the authors.

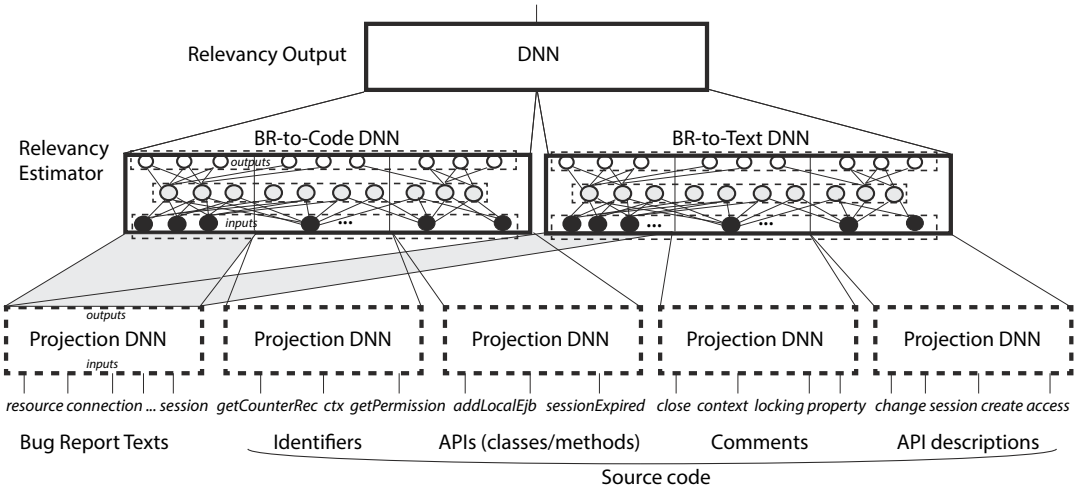

Fig. 2: DNNs for Projection and Relevancy Estimation

TABLE I: Subject Projects [7]

Project	Time Range	#BRs	#SourceFiles
AspectJ	03/02-01/14	593	4,439
Birt	06/05-12/13	4,178	6,841
Eclipse UI	10/01-01/14	6,495	3,454
JDT	10/01-01/14	6,274	8,184
SWT	02/02-01/14	4,151	2,056
Tomcat	07/02-01/14	1,056	1,552

For comparison, we used the same procedure as in Ye *et al.* [7]. Note that, the number of all source files in a project is very large, making the training time with all negative samples impossible. Thus, for training, they (and we) used text similarity measure to rank all the files, and selected only the top 300 similar files to the bug report as the negative samples. For prediction, we still compute the scores and rank all the files in a project. We sorted the bug reports chronologically by their report timestamps. For all projects except the smallest project AspectJ, we divided the bug reports into 10 folds with equal sizes in which $fold_1$ is the oldest and $fold_{10}$ is the newest. We trained a model on $fold_i$ and tested it on $fold_{i+1}$. For AspectJ, we divided the bug reports into 3 folds since it has smaller number of reports, and used the same testing strategy.

We used three metrics for evaluation. **Top-ranked accuracy** is measured as follows. If one of the buggy files of a given bug report is within the top-k list of files, we count it as a hit. Otherwise, we consider that as a miss. The top-k accuracy is measured by the percentage of hits over the total number of suggestions in all the folds. To consider the cases of a bug report with multiple buggy files, we also measured Mean Average Precision (**MAP**) and Mean Reciprocal Rank (**MRR**).

B. Impacts of Components and Parameters on Accuracy

In this experiment, we evaluated the impact of different components and parameters of HyLoc on its accuracy. We chose Tomcat, one of the smaller subject systems in our dataset.

1) Accuracy with Different Components: In this experiment, we varied different components and compared the accuracy of

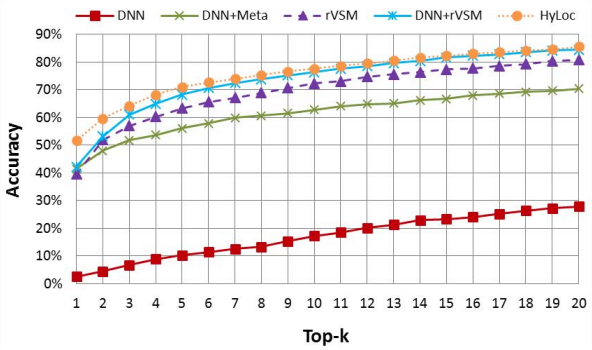

Fig. 3: Top-k Accuracy with Different Components

newly configured models to learn their impacts on accuracy. Figure 3 shows the result. The lines marked with DNN shows for the accuracy of the model using only DNN to compute bug-report-to-file relevancy as the sole feature. As seen, DNN by itself does not give high accuracy. Investigating further, we see that due to projection to a new space with smaller dimensions, more source files with the same technical topics are included but they are not buggy with respect to a given bug report. When using the relevancy feature via DNN and the bug-fixing metadata features (line DNN+Meta), the accuracy is much improved (top-1 accuracy from 3% to 41%, and top-5 accuracy from 10% to 56%). The model using text similarity (rVSM) is equivalent to BugLocator [10] (i.e., without using DNN and bug-fixing metadata features). The top-ranked accuracy ranges from 36–71%. However, when we combined relevancy feature computed by DNN and textual similarity feature computed by rVSM (see line DNN+rVSM), the accuracy is higher than those of both DNN and rVSM individually. The absolute improvement values over DNN are from 39–60% for top-ranked accuracy. The corresponding improvement values over rVSM are from 1–5% (relative improvement from 2.5–8%).

To study further the results from the DNN, rVSM, and DNN+rVSM models, we drew their venn diagrams (Figure 4). Our goal is to evaluate how much overlapping between the

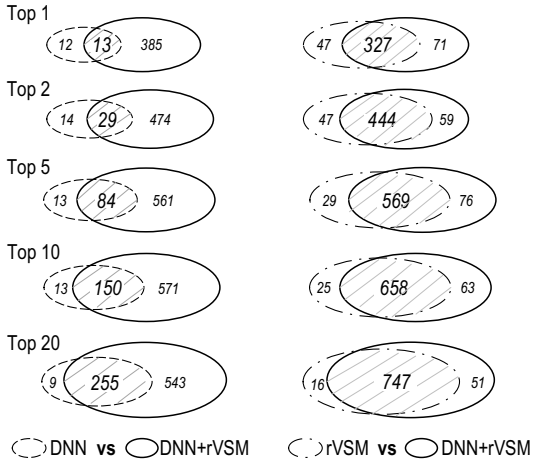

Fig. 4: Venn Diagrams for Correct Results of Approaches

correct result from the combining model DNN+rVSM and those from the individual models DNN and rVSM. We computed such overlaps for different top-ranked resulting lists (with 1,2,5,10, and 20 files). That is, if a buggy file was in a top-ranked list of a model, we consider that it is a correct case and vice versa. The left column shows the overlapping between the results of the model DNN and the combining model DNN+rVSM. For example, at the first diagram on the top left corner of Figure 4, for a top-1 list, in a total of 410 correct cases, DNN+rVSM is correct in 13+385=398 cases. Only 12 of them were correctly identified by DNN but not by DNN+rVSM, and 385 of them were correctly identified by DNN+rVSM, but not by DNN. In general, across all top ranks (1–20), the combining model covers a good percentage (52–96%) of the results by DNN, while it correctly identified many more results that DNN did not.

The second column of Figure 4 displays the overlapping between the correct result of rVSM and that of the combining model DNN+rVSM. It is consistent across all top-ranked accuracy that the correct result of DNN+rVSM covers a large percentage (88–98%) of that of rVSM. Moreover, the correct results of (DNN+rVSM $\setminus$ rVSM) are multiple times (1.3–3.2x) more than those of (rVSM $\setminus$ DNN+rVSM) (i.e., 2.1–6.9% of all total results). Interestingly, there are 18 cases in which both DNN and rVSM does not put the file in the top-20 list, but DNN+rVSM does.

Finally, HyLoc achieves highest accuracy with the combination of relevancy via DNN, textual similarity via rVSM, and the metadata features (Figure 4). With a single suggestion, it has the correct buggy file in almost 51% of the cases. Two out of three cases, a correct buggy file is in the list of 3 suggested files. With 5 suggested files, it is correct in 70% of the cases.

2) Example: We examined the cases where DNN contributes to improve accuracy while the IR model rVSM does not put the actual buggy files in a top-ranked list. They correspond to the cases in (DNN+rVSM $\setminus$ rVSM) (Figure 4) and/or DNN ranks the actual files higher in the resulting list than rVSM. We found a common phenomenon in those cases that reporters described in the bug reports the scenario(s) leading to failure or unexpected behaviors using texts without many code tokens. In those cases, the buggy files have few comments and do not contain many identifiers and terms that appear also in bug reports.

TABLE II: Linking Terms in Reports and Terms/Tokens in Files

BR	Token 1	Token 2	Token 3	Token 4
context	authorization	ctx	envCtx	asyncContext
resource	virtualClasspath	changeSessID	setSecureClass	addLocalEjb
writer	globalCacheSize	charset	index	charsWritten
read	headerLength	InternalBuffer	readBytes	dir

3) Examples of Linking Terms in Two Spaces: In NLP, researchers have shown that DNN is able to project and link the words that are semantically or grammatically related into nearby locations (at least along some dimensions) in a continuous-valued feature space. In this experiment, we study the examples in which HyLoc with its DNN machinery is able to learn the relations between the terms in the bug reports and the terms/tokens in the relevant source files.

We conducted an experiment on the DNN model in Figure 2 after training. We used each of the textual tokens in all bug reports as the input. We paired it with each of the code tokens in source files and the textual tokens in comments/documentations. We used that DNN to compute the score output for each of such pairs. Finally, for each textual token in a bug report, we produced the list of "relevant" tokens in source files ranked by the relevancy score between two tokens. Figure II displays a few examples of "related" terms in the bug report 25060 with top scores computed by the DNN model. As seen, despite using different lexical terms (e.g., context versus ctx and envCtx), DNN is able to relate them in two spaces. Moreover, the term resource does not appear in the buggy file StandardContext.java, however, DNN can relate it to the terms in that file such as virtualClasspath, changeSessID, setSecureClass, etc. since they appear frequently in the pairs of bug reports and buggy source files. Therefore, DNN is able locate and rank that buggy file highest.

4) Feature combination comparison: To compare feature combination by DNN and by learn-to-rank approach [7], we built another experimental model in which all three types of features in our model is combined via learn-to-rank, instead of DNN. We then compared the accuracy of that newly built model (called LRCombine) and the model where all features are combined via DNN. Our result shows that at top-1 accuracy, with DNN, our model achieves higher than LRCombine 22%, while the improvement at top-5 is 10%. This shows that the non-linear combination of those features is better than the linear one via learn-to-rank for this bug localization problem.

C. Accuracy Comparison

Our next experiment aims to compare HyLoc to the state-of-the-art approaches including the Naive Bayes (NB) approach by Kim *et al.* [5], the LR (learn-to-rank) approach by Ye *et al.* [7], and BugLocator by Zhou *et al.* [10]. While NB is an ML approach, BugLocator is IR-based, and LR is a hybrid one. We used the same dataset provided by Ye *et al.* [7] and conducted our experiment in the same procedure and metrics. Therefore, we used the results reported in the paper by Ye *et al.* [7] for comparison. In their paper, they also reported the result from running BugLocator on the same dataset. Thus, we also used that result for BugLocator. For the NB approach in Kim *et al.* [5] (which is not publicly available), we followed their description for re-implementation.

TABLE III: Accuracy Comparison

System	Model	1	2	3	4	5	10	15	20	MRR	MAP
TomCat	HyLoc	51.6	59.6	64.1	68.3	71.0	77.6	82.2	85.6	0.60	0.52
	LR	46.2	54.2	59.8	62.3	66.5	74.7	80.1	82.1	0.55	0.49
	BL	35.5	48.7	52.9	58.7	61.8	71.1	77.3	80.2	0.48	0.43
	NB	5.2	6.9	8.3	8.8	9.0	11.9	14.5	16.6	0.08	0.07
AspectJ	HyLoc	40.9	51.8	58.6	61.7	65.7	75.9	79.9	82.7	0.52	0.32
	LR	20.2	32.1	38.5	41.3	45.5	61.1	68.2	70.9	0.33	0.25
	BL	20.1	30.5	40.1	43.3	47.7	57.0	62.1	67.6	0.32	0.22
	NB	4.2	8.0	11.3	11.7	16.0	21.1	26.3	28.2	0.10	0.07
Birt	HyLoc	19.1	24.8	29.7	33.5	36.0	44.6	51.1	55.4	0.28	0.20
	LR	12.4	18.1	22.5	25.1	27.9	37.3	42.4	46.0	0.20	0.15
	BL	11.1	16.2	20.0	22.4	24.9	32.1	37.0	40.6	0.18	0.14
	NB	2.9	4.7	6.5	7.9	8.7	13.8	15.9	17.6	0.06	0.05
Eclipse	HyLoc	40.3	49.8	55.7	60.4	63.5	72.5	77.3	80.3	0.51	0.41
	LR	36.5	47.0	52.0	58.0	60.1	70.7	75.3	79.1	0.47	0.40
	BL	26.5	34.9	40.3	44.8	49.3	60.1	67.3	70.2	0.37	0.31
	NB	3.8	6.1	8.3	9.6	10.6	14.7	16.8	18.3	0.07	0.06
JDT	HyLoc	33.3	44.1	51.0	55.6	59.0	68.5	73.4	76.7	0.45	0.34
	LR	30.0	40.3	48.2	51.1	55.2	68.1	72.4	77.6	0.42	0.34
	BL	19.1	26.2	31.6	37.4	40.2	51.2	57.7	61.3	0.30	0.23
	NB	6.6	9.7	11.8	13.6	15.0	20.0	22.9	25.2	0.11	0.08
SWT	HyLoc	31.0	42.8	51.9	57.6	62.1	74.2	80.3	84.3	0.45	0.37
	LR	28.3	39.4	47.9	52.7	58.2	70.0	76.8	80.0	0.41	0.36
	BL	19.3	24.5	30.0	34.4	38.3	51.1	58.5	64.4	0.28	0.25
	NB	7.4	11.8	14.9	17.0	19.0	26.9	31.3	35.3	0.14	0.11

First, as seen in Table III, in comparison with BugLocator [10], at top-1 accuracy, HyLoc achieves from 8–20.8% higher (relatively 45–104% higher). At top-5 accuracy, the improvement is from 9.2–23.8%. As explained in Section IV-B1, the relevancy feature computed by the DNN model helps to complement the textual similarity feature computed by rVSM in BugLocator to bridge the lexical gap between bug reports and source files. Thus, HyLoc with an additional feature of project's metadata achieves even higher accuracy than BugLocator.

Second, compared to the learn-to-rank (LR) approach [7], at top-1 accuracy, HyLoc achieves from 2.7–20.7% higher (relatively 9.5–102.4% higher). At top-5 accuracy, the improvement is from 3.4–20.2%. The key to bridge the lexical gap in the LR approach is to add to the feature set the terms in the API documentation for the APIs used in the source files. We found that such addition is not effective in the cases of popular APIs such as java.util in Java Development Kit. Those APIs do not help in linking a bug report to source code because the report describes a more specific erroneous functionality in the system. In contrast, the DNN-based relevancy estimator can handle the lexical mismatch in those cases since it connects a report to a buggy file if the terms appear frequently enough in the pairs of bug reports and files. Third, we found that HyLoc is able to correctly suggest several cases where NB approach in Kim *et al.* [5] missed because their approach has not seen the fixed files before. The key reason is that their approach uses the previously fixed files as labels for training, but does not consider their contents. At top-1 amd top-5 accuracy, HyLoc achieves from 16.2–46.4% and 27.3–62% higher, respectively.

Generally, HyLoc also consistently achieves highest accuracy in MRR and MAP. Its MRR values are from 0.28–0.6. That is, in the best scenario, among 3 cases, it would rank an actual buggy file at the 2nd place in two cases, and likely rank another actual buggy file as the top candidate in the other case.

D. Time Efficiency

Table IV shows HyLoc's training and predicting time. All experiments were run on a computer with CPU Intel Xeon CPU E5-2650 2.00GHz (32 cores), 126 GB RAM. Since we need only one fold for training to predict the next fold, we measured the training time for one fold. We measured the predicting time for individual bug reports. As expected, training time is large for a solution involving one thread to run DNN. However, we could investigate parallel computing infrastructures for DNNs or the incremental training techniques to update the model after getting more bug reports and fixed files over time. Predicting time is reasonable (within a few minutes for one bug report).

TABLE IV: Training and Predicting Time in Minutes

System	Training for one fold		Predicting for one report	
	Max	Average	Max	Average
Tomcat	70	65	1.5	1.0
AspectJ	70	70	4.1	2.4
Birt	98	84	4.5	3.1
Eclipse	120	90	3.8	2.1
JDT	122	94	4.8	3.3
SWT	95	83	2.4	1.8

V. Conclusion

This paper presents a combining approach between *rVSM, an information retrieval technique*, and *deep neural network* (DNN) in which DNN is used to learn to connect the concrete terms in bug reports to the code tokens and terms in source files. Our empirical evaluation on several real-world projects shows that DNN and IR complement well to each other to achieve higher bug localization accuracy than individual models. Finally, our new model, HyLoc, achieves higher accuracy than state-of-the-art IR and machine learning techniques.

Acknowledgment

This work was supported in part by the US NSF grants CCF-1518897, CNS-1513263, CCF-1413927, CCF-1320578, CCF-1349153, TWC-1223828, CCF-1018600, and CCLI-0737029.

References

[1] E. Arisoy, T. N. Sainath, B. Kingsbury, and B. Ramabhadran. Deep neural network language models. In *Proceedings of the NAACL-HLT 2012 Workshop*, WLM '12, pages 20–28. ACL, 2012.

[2] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In ICSE '10, pages 95–104. ACM, 2010.

[3] Y. Bengio. *Foundations and Trends in Machine Learning - Learning Deep Architectures for AI*. NOW, the essence of knowledge, 2009.

[4] J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In ASE'05, pp. 273–282. ACM.

[5] 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*, 39(11):1597–1610, 2013.

[6] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Bug localization using latent dirichlet allocation. *Inf. Softw. Technol.*, 52(9):972–990, 2010.

[7] X. Ye, R. Bunescu, and C. Liu. Learning to rank relevant files for bug reports using domain knowledge. In FSE'14, pp. 689–699. ACM, 2014.

[8] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen. A topic-based approach for narrowing the search space of buggy files from a bug report. In ASE '11, pp. 263–272. IEEE, 2011.

[9] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, V. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and info. retrieval. *IEEE TSE*, 33(6):420–432, 2007.

[10] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports. In ICSE '12, pages 14–24. IEEE Press, 2012.