Presented at 2019 IEEE 16th International Conference on Mining Software Repositories in Montréal, QC, Canada.
Unfortunately, the manuscript as originally published in the MSR proceedings has crucial typos in Equations (5) and (6).
This version contains the correct versions of those equations and corrections for a few other minor typos.

SCOR: Source Code Retrieval With Semantics and Order

Shayan A. Akbar
Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
sakbar@purdue.edu

Avinash C. Kak
Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
kak@purdue.edu

***Abstract*—Word embeddings produced by the word2vec algorithm provide us with a strong mechanism to discover relationships between the words based on the degree to which they are contextually related to one another. In and of itself, algorithms like word2vec do not give us a mechanism to impose ordering constraints on the embedded word representations. Our main goal in this paper is to exploit the semantic word vectors obtained from word2vec in such a way that allows for the ordering constraints to be invoked on them when comparing a sequence of words in a query with a sequence of words in a file for source code retrieval. These ordering constraints employ the logic of Markov Random Fields (MRF), a framework used previously to enhance the precision of the source-code retrieval engines based on the Bag-of-Words (BoW) assumption. The work we present here demonstrates that by combining word2vec with the power of MRF, it is possible to achieve improvements between 6% and 30% in retrieval accuracy over the best results that can be obtained with the more traditional applications of MRF to representations based on term and term-term frequencies. The performance improvement was 30% for the Java AspectJ repository using only the titles of the bug reports provided by iBUGS, and 6% for the case of the Eclipse repository using titles as well as descriptions of the bug reports provided by BUGLinks.**

***Index Terms*—Source code search, word embeddings, information retrieval, bug localization**

I. Introduction

Representing textual words with their real-number based embeddings has emerged as a powerful approach for associating context-based meanings with the words and for comparing words for their similarity on the basis of such meanings. What is perhaps the most commonly used algorithm today for creating such embeddings is the word2vec algorithm proposed by Mikolov et al. [1].

Given the semantically rich representation for the words created by, say, word2vec, one is led to wonder if a retrieval framework based on such representations can be made even more powerful if it is subject to term-term ordering constraints modeled by, say, Markov Random Fields (MRF).

As we demonstrate in this paper, the answer to the question posed above is a categorical yes.

The retrieval framework that we present in this paper for establishing the above claim invokes MRF based ordering constraints on the query terms and the file terms that are matched on the basis of contextual semantics using the word2vec generated numeric vectors for the terms. This is facilitated by two "layers" that we refer as the "Match Layer (ML1)" and the "Match Layer 2 (ML2)". In the form of a 2D numeric array, ML1 is simply a record of the similarities between the terms in a query and the terms in a file, with the similarities being computed by applying the Cosine distance measure to the numeric vectors produced by word2vec. Subsequently, in the spirit of convolutional neural networks, we convolve the ML1 layer with a 2×2 kernel — whose elements must possess certain pre-specified properties — to yield another 2D numeric array that is ML2. As we argue in this paper, the numbers in the ML2 layer become high only for those sequences of terms in the query and a file, which is being evaluated for retrieval vis-a-vis the query, when there is significant simiarlity between the two both respect to the terms and with respect to the ordering constraints on the terms. As we show in this paper, convolving the 2D arrary of numbers in ML1 with a 2×2 operator produces the same effect as what would be achieved with MRF based logic as presented in [2], [3].

We have established the superiority of the "semantics plus order" approach for source-code retrieval over the more traditional methods by comparing the following retrieval frameworks in the context of automatic bug localization:[1] (1) the best-known approach based on the BoW (Bag-of-Words) assumption; (2) an approach based on MRF modeling using term and term-term frequencies; (3) a retrieval framework that uses contextual semantics through the word embeddings produced by the word2vec algorithm; and, finally, (4) a framework that uses MRF modeling on top of the word embeddings produced by the word2vec algorithm.

While the first three retrieval methods in the comparison mentioned above refer to frameworks that are already well known, the last — which combines MRF with the word embeddings produced by word2vec — is something that has

[1]Automatic bug localization is convenient for testing source-code retrieval algorithms on account of the relative ease with which ground-truthed datasets can be constructed for testing the performance of such algorithms. Nonetheless, it can be argued that, in general, source-code retrieval is a larger problem and not all of our conclusions may be applicable to it.

not been attempted before. Combining MRF with word embeddings allows us to jointly model the semantic and the ordering relationships in a single source code retrieval framework.

For our experiments, we use approximately 4000 bug reports of the Eclipse software library obtained from the BUGLinks dataset and approximately 300 bug reports of AspectJ obtained from the popular iBUGS dataset. We report results based on retrievals with (1) just the titles of the bug reports as queries; and (2) the entire description of the bug reports as queries. With both software libraries, we show that the retrieval precision can be improved between 6% and 30% over the best results that can be obtained with the more traditional applications of MRF to the representations based on term and term-term frequencies.

Comparing our SCOR (Source COde Retrieval) model with the best of what the literature has to offer, it significantly outperforms the purely MRF based framework presented in Sisman et al. [3]. SCOR also outperforms the more advanced BoW based techniques — BugLocator [4], BLUiR [5], and SCP-QR [6]. We also compare our retrieval model with semantic embeddings based bug localization algorithms — LSA (Latent Semantic Analysis) [7], and Ye et al's semantic retrieval algorithm [8]. Our results show that our retrieval algorithm can significantly outperform the LSA algorithm presented in [7], while the performance of our retrieval algorithm with respect to Ye et al's [8] semantic retrieval algorithm is comparable in terms of retrieval precision.

The work we report in this paper also involves training the word2vec model on what is perhaps the largest source code dataset ever put together; it consists of approximately 35000 Java repositories downloaded from GitHub. We learned word vectors for approximately 0.5 million software-centric terms from 1 billion word tokens. The reason for generating word vectors for such a large software vocabulary is to ensure that our semantic word embeddings are sufficiently generic so that they can be applied to new software repositories that were not used for generating the embeddings. Our results demonstrate that to be the case.

With this introduction the rest of the paper is organized as follows. In the "Related Works" section that follows, we briefly review some of the more prominent past investigations in bug localization. In section III we explain the word2vec model we used for constructing semantic word embeddings from large software corpora. In section IV we explain our novel retrieval model. We present our experimental results in section V and, finally, conclude in section VI.

II. Related Works

Today there exist several BoW based source code retrieval methods [4]–[7], [9]–[12]. However, since all of these methods are based on the simple bag-of-words assumption, they only consider the frequencies of the individual query terms in the source-code files. In other words, these methods do not take into account the ordering and the semantic relationships between the terms.

As it turns out, it is possible to enhance the BoW based methods with the modeling approaches developed by the researchers in the IR community [2], [13]–[15] that allow for the retrieval decisions to factor in the inter-term positional and ordering relationships. The earliest and the most popular of these dependency models, by Metzler and Croft [2], uses the notion of Markov Random Fields (MRF) to generate a second-order probabilistic model for a corpus, which is in contrast to the first-order probabilistic models that are generated by the BoW assumption. Sisman et al. [3] used the basic MRF modeling proposed by Metzler and Croft [2] and combined it with a *query conditioning* (QC) technique appropriate to the software context to report large improvements in retrieval precision. Note that the retrieval framework proposed in Sisman et al. [3] does not model the semantic relationship between the terms.

The notion of generating corpus-based semantic embeddings to model semantic relationships in a retrieval framework dates back to 1990's when Latent Semantic Analysis (LSA) was first published. Since then, LSA has successfully been applied to solve the problem of software search [7], [16].

The use of neural networks to learn semantic word embeddings was first proposed by Bengio et al. [17], with several modifications made to their neural network architecture by the contributions in [1], [18], [19]. From amongst these contributions, the word2vec implementation of [1] is auguably the most successful and the most commonly used today.

Several authors have investigated using word2vec in different application domains [20]–[24] that include web search, questing-answering systems, and paraphrase identification. Authors have also reported using word2vec for software search [8], [25]–[27]. These contributions, however, do not include ordering constraints.

Finally, note that researchers have also proposed using deep-learning based frameworks for solving the IR problem [28]–[33]. In general, such techniques, when applied to the domain of software search, require large training datasets that may not always be available. For example, the deep-learning based method reported by Lam et al. [34] used a training dataset consisting of around 20,000 bug reports. Other similar contributions are by Gu et al. [35] and Xiao et al. [36].

III. Constructing 0.5 Million Word Embeddings for Software-Centric Terms from 1 Billion Word Tokens

In this section we discuss the construction of neural word embeddings from large corpora of software repositories using the word2vec model [1], [37]. Word2vec is based on a single-layer neural network and it gives us a vector space that holds contextually semantic relationships between the words in a vocabulary. By "semantic relationships" we mean that the numeric vectors for the words that are contextually related should appear close together in the vector space. Two different words are contextually related if the words in their contexts are contextually related. The recursion implied by this definition is grounded by the statement that two words are contextually

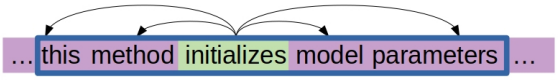


Fig. 1: An illustration of a context window of size 5 around the target term "initialize". The four terms "this", "method", "model", and "parameters" are the context terms of the target term "initialize" and will be used to train the model.

related if there exist common words in their respective contexts.

We trained the word2vec neural network by scanning each of the approximately 35000 Java source code repositories downloaded from GitHub for the training samples. Each training sample consists of a term in a repository along with a list of its context terms. The context terms are all the terms that appear in a context window whose width is a user defined parameter. The term around which a window is placed is called the "target" term, while the terms appearing inside the window other than the target term are called the "context" terms with respect to that target term. Figure 1 illustrates a context window.

The training pairs consisting of target and context terms are used to train the neural network to predict either the context terms from the target terms or the target terms from the context terms. The word2vec model, therefore, comes in two different flavors depending on whether the neural network predicts target terms from a given set of context terms — this is referred to as a CBoW (Continuous Bag Of Words) model, or predicts the context terms from a target term — this is called a skip-gram model. After training finishes, the learned weights in the neural network are used to construct meaningful vector representations of the terms in a vocabulary set.

In our retrieval experiments we use the skip-gram flavor of word2vec which we discuss in the subsection that follows. However, note that the CBoW model can also be used to produce similar results. The skip-gram and CBoW word embeddings are available online [38].

A. *word2vec for Constructing a Skip-Gram*

As mentioned above, for a skip-gram, the word2vec neural network is designed to predict the context terms for a given target term. The context terms are the terms that can be expected to appear with high likelihood in a small window around the relevant target term. The neural network, shown in Figure 2, consists of three layers: input, projection, and output. The nodes in the adjacent layers are fully connected.

In a skip-gram neural network, the sizes of the input and the output layers are equal to the size V of the vocabulary, while the size of the projection layer in the middle is N — a user set parameter. The parameter N actually controls the dimensionality of the word vectors that we want to learn from the neural network as we explain later in this section. The terms in the vocabulary are arranged in alphabetical order and each term assigned the index that corresponds to its position in the vocabulary.

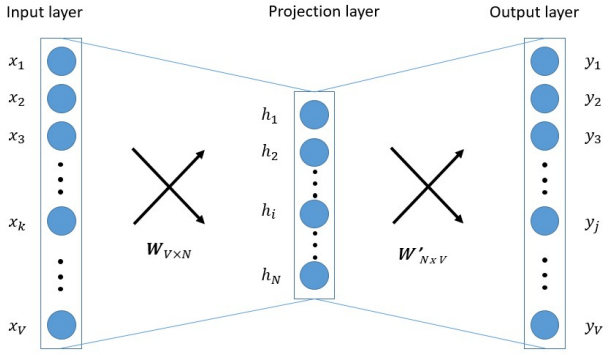


Fig. 2: The neural layout for the skip-gram constructor. A skip-gram predicts the context terms for given target terms. The one-hot encoding of context terms are provided at input, while the softmax probabilities of terms in the vocabulary are computed at output.

The input to the neural network is a V dimensional one-hot encoding vector for a term. One-hot encoding consists of a single entry of 1 at the position of the term in the alphabetized vocabulary listing, with all the other entries set to 0. As mentioned earlier, the output layer of the neural network also consists of V nodes. In the V-dimensional output each node represents the probability of prediction for each term in the vocabulary.

The input layer is commonly represented by the vector $\mathbf{x} = \{x_1, ..., x_V\}$, the projection layer by the vector $\mathbf{h} = \{h_1, ..., h_N\}$, and output layer by the vector $\mathbf{y} = \{y_1, ..., y_V\}$. The weights between the input and projection layers are denoted by a $V \times N$ matrix $\mathbf{W}$. Each row of $\mathbf{W}$ is the N-dimensional *input vector* $\mathbf{v}_w$ for the term w in the vocabulary. This follows from the fact that if we represent the one-hot vector for a target term by w_I, the projection layer output would be given by $\mathbf{h} = \mathbf{W}^T\mathbf{x} = \mathbf{v}_{w_I}$, which is the I^{th} row of the matrix W.

We also have another matrix $\mathbf{W}'$ with dimensions $N \times V$ for the weights between the projection layer and the output layer. Each column of $\mathbf{W}'$ is an N-dimensional *output vector* representation $\mathbf{v}'_w$ for the term w. As a result of applying the weights in $\mathbf{W}'$ to the outputs of the projection layer we end up with a net score u_j for each term j in the vocabulary $u_j = \mathbf{v}'^T_{w_j}\mathbf{h} = \mathbf{v}'^T_{w_j}\mathbf{v}_{w_I}$, where $\mathbf{v}'^T_{w_j}$ denotes the j-th column of $\mathbf{W}'$. The net score u_j for a term w_j is equivalent to the product of the input vector $\mathbf{v}_{w_I}$ of the target term w_I (provided at the input) and the output vector $\mathbf{v}'_{w_j}$ of the term w_j we want to predict at the output.

The softmax nonlinearity function is applied to the net scores at the output layer to convert them into probability estimates $p(w_j|w_I)$:

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V}\exp(u_{j'})} = \frac{\exp(\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I})}{\sum_{j'=1}^{V}\exp(\mathbf{v}'^T_{w_{j'}}\mathbf{v}_{w_I})} \quad (1)$$

Notice that y_j is the output of j-th unit of output layer and represents the probability that term w_j is the context term given the target term w_I at input. We want to maximize the above conditional probability $p(w_j|w_I)$.

Rong [39] provides a detailed explanation of how the parameters of a word2vec skip-gram neural network are learned using the backpropagation algorithm with stochastic gradient descent and a "negative sampling" efficiency optimization trick [1], [37]. After the learning process converges, the resulting vectors $\mathbf{v}_w$ and $\mathbf{v'}_w$ correspond to the terms w in the vocabulary. In our retrieval experiments, we discard the output vectors $\mathbf{v'}_w$ and use only the input vectors $\mathbf{v}_w$ as the vector representations of terms in the vocabulary. We refer to each vector $\mathbf{v}_w$ as the semantic embedding for the term w. The contextual semantic relationship between any two terms w_1 and w_2 may be obtained by comparing the vectors $\mathbf{v}_{w_1}$ and $\mathbf{v}_{w_2}$ using an appropriate metric.

IV. Modeling Ordering and Semantic Relationships for Software Retrieval

In this section we present how we jointly model the two seemingly disparate aspects of our framework for source code retrieval: the term-term ordering constraints and the word2vec generated semantic relationships between the terms. The term-term ordering constraints are imposed using the MRF framework described by Sisman et al. [3], whereas the semantic relationships between the terms are modeled by comparing the semantic word embeddings [1] of the query and the file terms using cosine similarity measure.

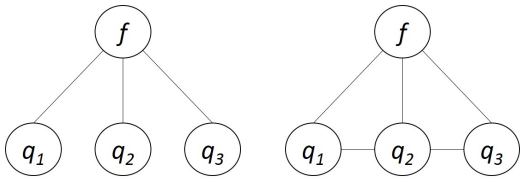
A. Modeling Ordering Relationships Between Terms

In the context of IR based bug localization, a Markov Random Field is an undirected graph G in which one of the nodes represents a source-code file f that is being evaluated for its relevance to a given query Q and all other nodes represent the individual terms $Q = \{q_1, q_2, ..., q_{|Q|}\}$ in the query. The arcs between the nodes represent probabilistic dependencies between the nodes [3].

The MRF framework gives us the liberty to choose different kinds of probabilistic dependencies we want to encode in the retrieval model. Figure 3 shows two possible dependency assumptions that we can make about the construction of the MRF graph G. One is called the "Full Independence" (FI) assumption in which all the query terms are independent of one another. Notice the absence of arcs between nodes that stand for the query terms q_1, q_2, and q_3. The other is called the "Sequential Dependence" (SD) assumption in which the nodes for the consecutive query terms are connected to each other via arcs as shown. We use the SD model to incorporate term-term ordering relationships between the query terms that may be present in the file f.

The Dirichlet smoothed BoW based Full Independence (FI) relevance score is calculated as:

$$score_{fi}(Q, f) = \sum_{i=1}^{|Q|} \log \frac{tf(q_i, f) + \frac{\mu_{fi} tf(q_i, C)}{|C|}}{(|f| + \mu_{fi}) \frac{tf(q_i, C)}{|C|}} \qquad (2)$$



(a) Full Independence (b) Sequential Dependence

Fig. 3: Using a three-term query as an example, illustrated here are the two different inter-term dependency assumptions for MRF modeling of a software library.

where $tf(q_i, f)$, and $tf(q_i, C)$ are, respectively, the frequencies of the term q_i in the source code file f and in the collection C. The notations $|f|$ and $|C|$ stand for the size of the file and the collection, respectively. Finally, μ_{fi} is the smoothing parameter.

Denoting a pair of sequential terms $q_i q_{i+1}$ by ρ, the MRF SD model gives us the following formula for the relevance of a file f to a query Q:

$$score_{sd}(Q, f) = \sum_{i=1}^{|Q|-1} \log \frac{tf_w(\rho, f) + \frac{\mu_{sd} tf_w(\rho, C)}{|C|}}{(|f| + \mu_{sd}) \frac{tf_w(\rho, C)}{|C|}} \qquad (3)$$

where $tf_w(\rho, f)$ and $tf_w(\rho, C)$ are, respectively, the frequencies of the pair of terms $\rho = q_i q_{i+1}$ in a file f and in the collection C. The notation μ_{sd} is for the smoothing parameter.

B. Modeling Semantic Relationships Between the Terms

With the help of Figure 4, we now illustrate with an example our semantic retrieval framework that includes ordering constraints. The example used in the figure assumes that the query consists of just four terms. That is, $Q = \{q_1, q_2, q_3, q_4\}$. And that a file being evaluated for its relevance to the query consists of just five terms. That is, $f = \{t_1, t_2, t_3, t_4, t_5\}$. We also assume that the dimensionality of the numeric term vectors provided by word2vec is N. As we explain later in Section III, we will obtain these vectors by training a word2vec skip-gram model on a large corpus of Java source code repositories.

As shown in the figure at its left, the numeric vectors for the query terms and the file terms serve as inputs to the processing chain in Figure 4. The query terms q_i and the file terms t_j are compared pairwise using the cosine similarity measure shown below to produce the "Match Layer 1" (ML1):

$$\sigma_1(\mathbf{v}_{q_i}, \mathbf{v}_{t_j}) = \frac{\mathbf{v}_{q_i} \cdot \mathbf{v}_{t_j}}{||\mathbf{v}_{q_i}|| ||\mathbf{v}_{t_j}||} \qquad (4)$$

Each row of ML1 corresponds to the cosine similarity between a given query term q_i and all the terms in the file f. The layer ML1 is then treated as described below to produce a *semantic* relevance score.

As shown in the top row of Figure 4, we take the maximum of ML1 across all the file terms to produce the "Best-matching vector". In this vector, we retain only the largest cosine similarity value for each query term that corresponds to its best matched file term. The cosine similarity values for all

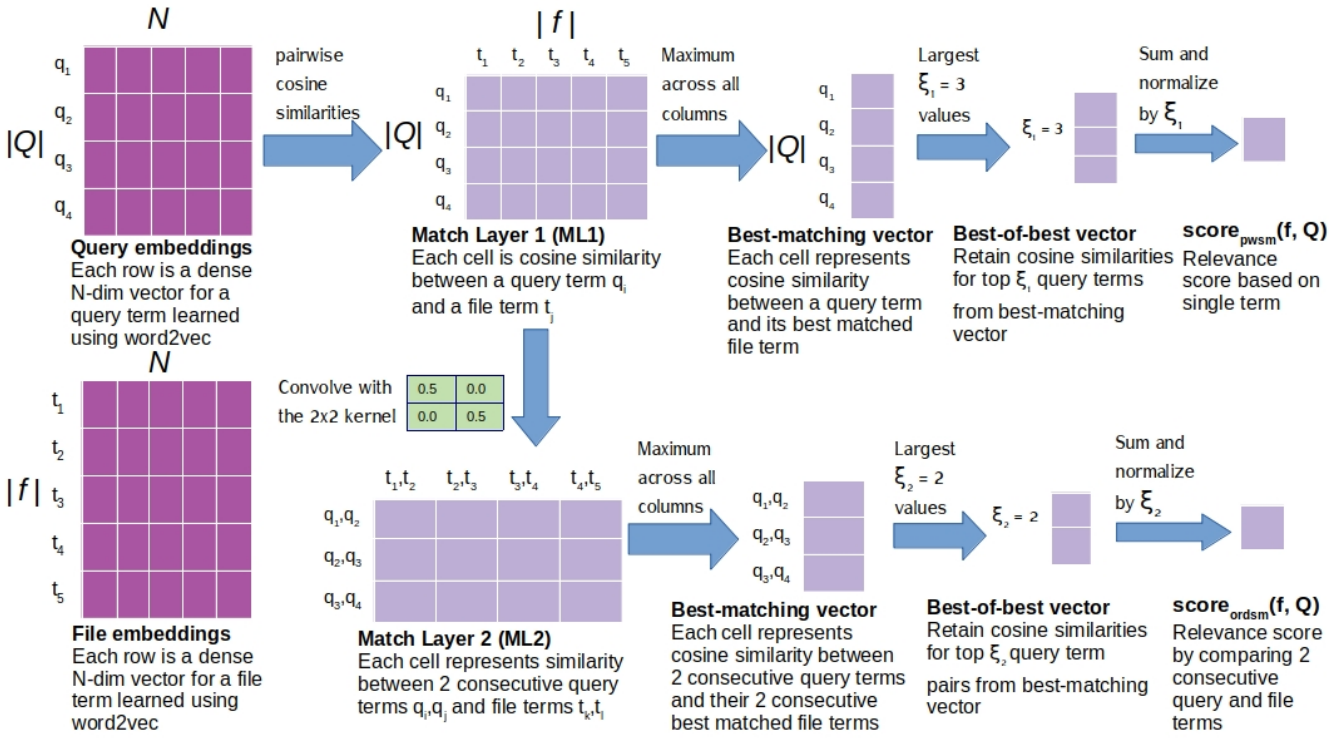


Fig. 4: **An illustration of a semantic retrieval framework with ordering constraints:** The N-dimensional numeric vectors for the terms in a query Q and a file f are provided as inputs as shown at left. The query terms and file terms are compared pairwise using cosine similarity to produce ML1. As shown in the top row, we perform further processing on ML1 to produce relevance score based on matching terms individually $score_{pwsm}(f, Q)$. As shown in the bottom row we convolve ML1 with a pre-defined 2×2 kernel to produce ML2, which is subsequently processed to produce the relevance score $score_{ordsm}(f, Q)$.

other file terms are discarded and, therefore, do not influence the final relevance score.

Afterwards, only the largest ξ_1 values from the "Best-matching vector" are retained. This second maximum selection operation discards the cosine similarity values for all those query terms that are deemed to not match the file sufficiently. In this manner, the terms that may negatively influence the final relevance score are dropped.

We refer to the vector obtained by the second maximum operation as the "Best-of-best vector" since it is obtained as a result of two successive maximum operations on array ML1 of cosine similarity values: (1) The first maximum is taken across all file terms for each query term to produce the best cosine similarity value for each query term that corresponds to its best-matched file term, (2) The second maximum is taken across all the query terms to produce a vector $\Omega(q_i)$ of cosine similarity values for only those ξ_1 query terms $q_i \in Q'$, where $Q' \subseteq Q$, that best match the file. The parameter ξ_1 is tunable.

The retained ξ_1 cosine similarity values are summed and normalized to obtain the relevance score $score_{pwsm}(f, Q)$ based on our Per-Word Semantic Model (PWSM):

$$\frac{1}{\xi_1} \sum_{q_i \in Q'} \Omega\Big(q_i \in \underset{\substack{Q' \subseteq Q \\ |Q'| \leq \xi_1}}{\mathbf{argmax}} \sum_{q_i \in Q'} \max_{t_j \in f} \sigma_1(\mathbf{v}_{q_i}, \mathbf{v}_{t_j})\Big) \tag{5}$$

where Q' is a subset of query Q composed of only those ξ_1 query terms for which we get the largest cosine similarities.

The great thing about the array of numbers in ML1 is that it lends itself to further processing that allows the incorporation of the MRF based term-term ordering constraints in how a query is matched with a file.

As shown in the bottom row of Figure 4, we now convolve the array of numbers in ML1 with a pre-defined 2×2 kernel K to produce another layer "Match Layer 2" (ML2). Each element in the array of numbers in ML2 represents the similarity measure between two consecutive query terms $q_i q_{i+1}$ and two consecutive file terms $t_j t_{j+1}$. This obviously implies incorporating ordering relationships between the query terms that semantically match the file terms.

The 2×2 kernel K mentioned above is designed such that it has non-zero values on its diagonal, while its off-diagonal elements are either zero or very close to zero. This is a key condition on K that allows the ordering constraints to be satisfied as explained below.

During the convolution of K with the array of numbers in ML1, the output at a specific location (i, j) results in a high value if the cosine similarity values between the terms q_i and t_j, and terms q_{i+1} and t_{j+1} are both high. The convolution operation, which results in a weighted sum over 2×2 neighborhoods in ML1, is similar to what is used in modern deep convolutional neural networks. The similarity

TABLE I: Stats related to the two different software libraries used for retrieval experiments: Eclipse and AspectJ.

	Eclipse	AspectJ
Description	IDE	Java extension
Programming Language	Java	Java
Number of bug reports	4035	291
Average number of relevant files report	2.76	3.09
Number of bug reports with stack traces	519	89
Number of bug reports with patches	8	4

values σ_2 in the ML2 are computed from the similarity values σ_1 of the ML1 as follows:

$$\sigma_2(q_iq_{i+1}, t_jt_{j+1}) = K_{11}\sigma_1(\mathbf{v}_{q_i}, \mathbf{v}_{t_j}) + K_{22}\sigma_1(\mathbf{v}_{q_{i+1}}, \mathbf{v}_{t_{j+1}}) + K_{12}\sigma_1(\mathbf{v}_{q_i}, \mathbf{v}_{t_{j+1}}) + K_{21}\sigma_1(\mathbf{v}_{q_{i+1}}, \mathbf{v}_{t_j})$$

The array of numbers in ML2 are treated in the same manner as those in ML1 to produce the ordered-semantic (ORDSM) relevance score $score_{ordsm}(f, Q)$ based on comparing two consecutive query terms $\rho_i = q_iq_{i+1}$ with two consecutive file terms $e_j = t_jt_{j+1}$. We use P and E to represent sets of pairs of query terms and file terms, respectively. Notice that we use a new parameter ξ_2 that is different from ξ_1 to obtain the "Best-of-best vector" $\Delta(\rho_i)$ from the "Match Layer 2" (ML2). Here, $\rho_i \in P'$ are the ξ_2 pairs of query terms that produce the largest similarity values, and $P' \subseteq P$. The ordered-semantic (ORDSM) score is computed as:

$$\frac{1}{\xi_2} \sum_{\rho_i \in P'} \Delta\Big(\rho_i \in \mathbf{argmax}_{\substack{P' \subseteq P \\ |P'| \leq \xi_2}} \sum_{\rho_i \in P'} \max_{e_j \in E} \sigma_2(\rho_i, e_j)\Big) \quad (6)$$

C. Computing a Composite Score for a Repository File

The previous two subsections presented different formulas for measuring the relevancy of a file to a query. The formulas in Section IV-A showed how a file could be ranked vis-a-vis a query using just the BoW modeling of the relationship between the two and also using the MRF based ordering constraints. And the formulas in Section IV-B showed how a file could be ranked purely on the basis of the similarities of the term-contextual relationships and on the basis when ordering constraints are superimposed on top of the term-contextual relationships.

We now combine all those measures of relevancy of a file to query to create a composite file relevancy score. As shown below, this composite formula uses a weighted aggregation of the scores given by the Equations (2), (3), (5), and (6):

$$score_{scor}(f, Q) = \alpha \cdot score_{fi}(f, Q) + \beta \cdot score_{sd}(f, Q) + \gamma \cdot score_{pwsm}(f, Q) + \eta \cdot score_{ordsm}(f, Q)$$

where α, β, γ, and η are tunable parameters.

V. Experimental Results

With the framework we have presented in the preceding sections, we now present our experimental results on source-code retrieval for solving the problem of automatic bug localization.

TABLE II: Stats related to the large Java corpus that we used to learn the semantic word vectors from the word2vec model.

	Statistics
Number of repositories	34264
Programming Language	Java
Size of raw dataset	368 GB
Number of source code files	3444730
Number of word tokens	940053404
Number of words in vocabulary	415554

We report results using two different software libraries: Eclipse and AspectJ. The results for the Eclipse [40] and AspectJ [41] libraries are for two different types of retrievals: using just the titles of the bug reports as queries ("title-only"), and using the entire bug reports as queries ("title+desc").

The bug reports for Eclipse and AspectJ software libraries were obtained from the publicly available BUGLinks [42] and iBUGS [43] datasets, respectively. Table I presents the relevant stats related to the two software libraries. Eclipse is a Java based software with a large number of bug reports (4035) in the BUGLinks dataset. The iBUGS dataset, on the other hand, contains relatively smaller number of bug reports (291) for the AspectJ repository, also in Java.[2]

In what follows, we first present the overall processing pipeline of our bug localization framework. Then we describe the metrics used for evaluating the source code retrieval results. That is followed by a motivating example that illustrates the power of semantic modeling for retrieval. Lastly, we present our experimental results.

A. Overall Framework

Figure 5 shows the steps involved in our bug localization framework. The word2vec neural network shown at left takes for its input a very large corpus of Java source code repositories and generates semantic word embeddings for the software-centric words present in those repositories. Approximately 35000 open-source Java repositories were downloaded from GitHub [44] were used in this study. Except for a few that are now defunct, we downloaded the same repositories as those listed in [45]. Table II shows the statistics of our Java source code dataset.

We used the popular Gensim library [46] to learn the word embeddings from the Java source-code dataset. The two important parameters that we can tune for training the Skip-gram model are: (1) vector size (N), and (2) window size (w). We set $N = 1000$, and $w = 8$ for all our retrieval experiments. The word vectors learned from the Skip-gram model are stored in a disk file.

The word embeddings along with the source-code files for the library (such as Eclipse) that we are interested, and also the bug reports, are fed into the retrieval engine.

The source-code files go through Porter stemming and stop word removal. Subsequently, each source-code file is indexed,

[2]Although BugLinks dataset contains 4650 bug reports for Eclipse and iBUGS dataset 350 bug reports for AspectJ, we chose 4035 from the former and 291 from the latter for our analysis. We ignored the bug reports for which we could not find in the repositories any of the source code files mentioned in the bug reports.

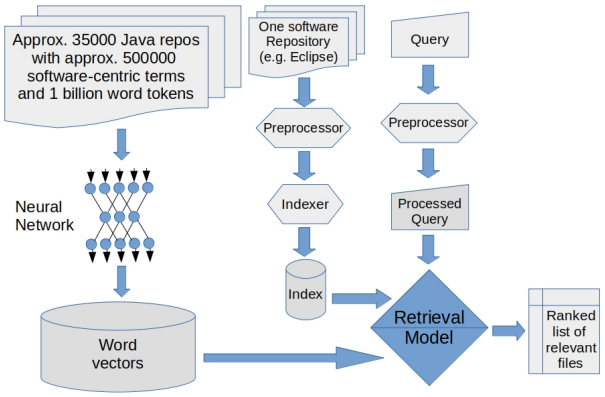


Fig. 5: Block diagram for the retrieval framework.

TABLE III: Summary of retrieval models along with the parameter settings that yielded the best results for each.

Method Name, Modeling scheme	Parameters Eclipse (title)	Parameters Eclipse (title+desc)	Parameters AspectJ (title)	Parameters AspectJ (title+desc)
SCOR, FI BoW + MRF SD + PWSM + ORDSM	$w = 8$ $\mu_{fi} = 1k$ $\mu_{sd} = 8k$ $\xi_1 = 10$ $\xi_2 = 3$ $\alpha = 0.32$ $\beta = 0.12$ $\gamma = 3.2$ $\eta = 25$	$w = 8$ $\mu_{fi} = 1k$ $\mu_{sd} = 4k$ $\xi_1 = 10$ $\xi_2 = 3$ $\alpha = 0.3$ $\beta = 0.12$ $\gamma = 2.5$ $\eta = 30$	$w = 8$ $\mu_{fi} = 3.5k$ $\mu_{sd} = 4k$ $\xi_1 = 10$ $\xi_2 = 5$ $\alpha = 0.3$ $\beta = 0.15$ $\gamma = 2.8$ $\eta = 25$	$w = 8$ $\mu_{fi} = 3.5k$ $\mu_{sd} = 4k$ $\xi_1 = 10$ $\xi_2 = 5$ $\alpha = 0.7$ $\beta = 0.04$ $\gamma = 2.8$ $\eta = 27$
MRF SD, FI + SD	$w = 8$ λ_{sd} = 0.2 $\mu = 4k$	$w = 20$ λ_{sd} = 0.2 $\mu = 4k$	$w = 2$ λ_{sd} = 0.2 $\mu = 4k$	$w = 16$ λ_{sd} = 0.2 $\mu = 4k$
PWSM, FI BoW + PWSM	$\mu_{fi} = 2.5k$ $\xi_1 = 10$ $\gamma = 0.95$	$\mu_{fi} = 4k$ $\xi_1 = 10$ $\gamma = 0.9$	$\mu_{fi} = 3.5k$ $\xi_1 = 10$ $\gamma = 0.95$	$\mu_{fi} = 3.5k$ $\xi_1 = 10$ $\gamma = 0.8$
FI BOW, Dirichlet smoothing	$\mu = 4000$	$\mu = 4000$	$\mu = 4000$	$\mu = 4000$

an inverted index constructed from the main index, and the two stored in hash tables. On the other hand, the bug reports are first subject to regular-expression based testing for the detection of stack traces or code patches. If these are absent, further testing is carried out for the detection of camel-cased source code identifiers. In the absence of such identifiers, the bug reports are subject to the same preprocessing steps as the source code files.

B. Evaluation Metrics

We use precision based metrics — specifically MAP (Mean Average Precision), $P@r$ for precision at rank r, and $R@r$ for recall at rank r — for a quantitative characterization of the performance of the different retrieval models. MAP and $P@1$ would generally be considered to be the most important metrics for evaluating the power of retrieval algorithms of the type under investigation here [47].

In order to determine whether the improvement observed in the retrieval results obtained using one model vis-a-vis another is significant, we carried out significance testing based on the Student's Paired t-Test [48].

C. A Motivating Example

In this section we use a simple retrieval exercise to show the power of semantic modeling in a source code retrieval system. We use the title of a bug report with ID 106140 filed for the Eclipse software library as a query to the two retrieval models — FI-BOW and PWSM — and compare their results. For reasons of space limitations we ignore the description of the bug that is a part of the bug report.

The title of bug ID 106140 reads "[compiler] Eclipse3.1.0: unrecognized class invisibility", which after preprocessing produces the term tokens ["compiler", "eclipse", "unrecognized", "class", "inivisibility"]. The Eclipse source code files that were fixed in response to this bug are:

1) org.eclipse.jdt.core/compiler/../lookup/Scope.java
2) org.eclipse.jdt.core.tests.compiler/../LookupTest.java

Examining the top-ranked 100 retrievals, while the average precision using the FI-BoW model for this bug report is just 0.0, when semantic relationships between terms are modeled using Per-Word Semantic Model (PWSM) the AP increases slightly to 0.03. What that means is that while FI BoW could not retrieve any of the two relevant files, the PWSM model could retrieve the "Scope.java" file.

Investigating further, we found that while the non-discriminatory query terms "compiler", "eclipse", and "class" did appear in the "Scope.java" source code, the important discriminatory terms "unrecognized", and "invisibility", which could rank "Scope.java" higher than the other files containing the non-discriminatory terms, did not. Examining the "Scope.java" more closely, we found that even though it did not contain the exact terms "unrecognized" and "invisibility", it did contain their semantically related terms "missed", and "visible". In fact, the term "visible" appeared approximately 100 times in the file. Therefore, when "Scope.java" is scored using PWSM, the semantic matching between terms "visible" and "invisibility" makes the overall score for the file higher than the other files which only contained the non-discriminatory query terms.

D. Retrieval Experiments

We provide experimental results for comparing the following four retrieval models: (1) FI BOW, (2) MRF SD, (3) PWSM, (4) SCOR. Table III displays the tunable parameter values used for each model. These parameters were set to yield the best performance from each model.

Through our experiments we attempt to answer the following important research questions (RQs):

RQ1: How good are the software-centric word vectors?
RQ2: Does the IR model that only incorporates ordering relationships improve the result over BoW IR models?
RQ3: Does the IR model that only incorporates semantic relationships improve the result over BoW IR models?
RQ4: How does our novel SCOR retrieval model perform against various BoW based source code retrieval models?
RQ5: How good is SCOR against pure MRF based retrieval techniques that only model ordering relationships?

TABLE IV: Some pairs of words and their abbreviations sampled from the SoftwarePairs-400 benchmark.

Abbr.	Word	Score	Abbr.	Word	Score
del	delete	5	med	median	3
tmp	temporary	5	col	column	5
rght	right	2	tot	total	4
min	minimum	5	acc	accept	1
num	number	5	alloc	allocate	5

TABLE V: Some words with their top 3 most (cosine) similar words as learned from the word2vec Skip-gram model.

rank	alexnet	delete	rotation	add	parameter
1	resnet	remove	angle	list	param
2	lenet	update	rot	set	method
3	imagenet	copy	lhornang	create	argument

RQ6: How does our SCOR retrieval model perform against other semantic embeddings based retrieval models?
RQ7: Are word2vec based word vectors generic enough to be used for searching in a new software library?

RQ1: How good are the software-centric word vectors?

To answer the question posed above we need to evaluate the quality of the word embeddings generated by the word2vec model for terms that occur in the software context. We, therefore, need a semantic similarity benchmark with which we can measure the quality of the word2vec generated word embeddings.

In this section we present an evaluation benchmark for word embeddings obtained for software-centric words. We also evaluate the software-centric word vectors obtained from word2vec model using our novel evaluation benchmark.

1) Semantic Similarity Benchmark: There exist in the natural language processing (NLP) research literature [49] many semantic similarity evaluation benchmarks for natural language words that are found in the articles of general interest, e.g. Wikipedia articles. There also exist domain-specific evaluation benchmarks for semantic word embeddings of a wide range of concepts in medicine, such as disease names, and medical procedures [50]. However, to the best of our knowledge, no such evaluation benchmark exists for software-centric word embeddings. Therefore, we present in this section for the first time a semantic similarity evaluation benchmark exclusively for software-centric word vectors.

In NLP and also in the literature related to medicine, what these human-created benchmarks contain are pairs of words

TABLE VI: Evaluation results on semantic similarity benchmark SoftwarePairs-400 for Skip-gram and CBoW models while changing N, which is the dimension of the word vectors.

Model (N)	C@1 (%)	C@5 (%)	C@10 (%)	Correlation
SG (1500)	105 (26%)	140 (35%)	161 (40%)	0.221
SG (1000)	112 (28%)	153 (38%)	172 (43%)	0.224
SG (500)	23 (5%)	52 (13%)	62 (15%)	0.108
SG (200)	19 (4%)	42 (10%)	53 (13%)	0.128
CBoW (1500)	38 (9%)	61 (15%)	69 (17%)	0.010
CBoW (1000)	45 (11%)	63 (15%)	67 (16%)	0.020
CBoW (500)	15 (3%)	37 (9%)	57 (14%)	0.053
CBoW (200)	19 (4%)	45 (11%)	53 (13%)	0.141

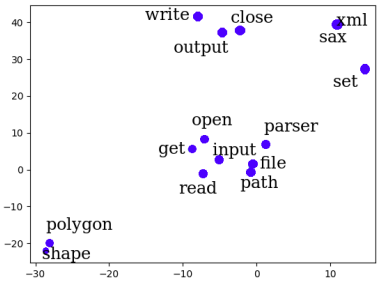


Fig. 6: Zoomed-in view of the word vectors in the first two dimensions. Note the clusters:: (“write”, “output”, “close”), (“open”, ”read”, “input”, “file”), (“polygon”, “shape”)

that are similar in meaning, and therefore, should have similar word vectors. For example, a pair could contain the words “female” and “woman”, or “tumor” and “cancer”, because they are semantically similar. These benchmarks also contain a human supplied semantic similarity score on a scale of 1 through 5 for each pair of such words. Therefore, to evaluate the word embeddings obtained from a model like word2vec, the word vectors of the semantically similar words in the list of pairs in the benchmark are compared against each other using a similarity measure, say the cosine similarity. Subsequently, these calculated similarity scores can be compared with the human-supplied scores to measure the quality of the embeddings.

With inspiration drawn from prior research in semantic word embeddings in NLP and medicine, we have created a benchmark called SoftwarePairs-400 for terms occuring specifically in the software programs. In our novel benchmark, we carefully compiled a list of pairs of 400 words along with their commonly used abbreviations in programming languages. A sample from this list of pairs along with their respective human-assigned scores is given in Table IV.

2) Evaluation: We now evaluate the word embeddings produced by word2vec for software-centric words using two evaluation metrics — correct @r ($C@r$), and Pearson correlation score. The correct at r ($C@r$) metric is defined as the number of pairs in the list of 400 pairs, in which the abbreviation appears in the top r ranked positions when the vector for the term corresponding to the abbreviation is compared with all the vectors in the database using cosine similarity. The higher the value for $C@r$ the better is the word embedding model. On the other hand, the Pearson correlation score is calculated by comparing the human-assigned scores to the pairs of words, with the cosine similarity values obtained when the vectors of words present in the pair are compared against each other. Higher correlation values imply better word embedding models.

In Table VI we provide evaluation results on SoftwarePairs-400 for two different models: (1) Skip-gram (SG), and (2) Continuous Bag of Words (CBOW). Both the models are run with four different values of N — 200, 500, 1000, or 1500. Our results show that the Skip-gram model with $N = 1000$ produces the best values for $C@r$. Therefore we run all our

TABLE VII: Retrieval accuracy for the "title-only" queries.

Method	Eclipse			
	MAP (%) (p-value)	P@1	P@5	R@10
SCOR	**0.2709 (32.8%)**$^{\mp *}$ (3e-20)	**0.238**	**0.115**	**0.332**
FI + MRF SD	0.2493 (22.2%)* (1e-10)	0.211	0.108	0.315
FI + PWSM	0.2336 (14.5%)* (2e-5)	0.192	0.101	0.297
FI BoW	0.2039	0.169	0.088	0.258
	AspectJ			
SCOR	**0.1802 (44.2%)**$^{\mp *}$ (8e-3)	**0.206**	0.092	0.193
FI + MRF SD	0.1348 (7.9%)* (6e-2)	0.134	0.079	0.167
FI + PWSM	0.1641 (31.3%)* (4e-2)	0.164	**0.094**	**0.197**
FI BoW	0.1249	0.137	0.070	0.149

$*$ significantly different from FI BOW.
$\mp$ significantly different from MRF SD.

TABLE VIII: Retrieval accuracy for the "title+desc" queries.

Method	Eclipse			
	MAP (%) (p-value)	P@1	P@5	R@10
SCOR	**0.3204 (29.1%)**$^{\mp *}$ (6e-20)	**0.289**	**0.134**	**0.394**
FI + MRF SD	0.3034 (22.2%)* (5e-31)	0.272	0.127	0.374
FI + PWSM	0.2713 (9.3%)* (1e-3)	0.233	0.116	0.341
FI BoW	0.2481	0.210	0.106	0.317
	AspectJ			
SCOR	**0.2506 (17.6%)**$^{\mp *}$ (1e-3)	**0.299**	**0.127**	**0.299**
FI + MRF SD	0.2263 (6.24%)* (2e-2)	0.264	0.114	0.265
FI + PWSM	0.2334 (9.5%)* (1e-2)	0.244	0.125	0.283
FI BoW	0.2130	0.244	0.105	0.243

$*$ significantly different from FI BOW.
$\mp$ significantly different from MRF SD.

retrieval experiments using the semantic word vectors learned with the Skip-gram model with $N = 1000$. It can also be observed from the experimental results that CBoW performs worse than Skip-gram. Also, the SG model with $N = 1500$ performs worse than SG model with $N = 1000$. Therefore, our results show that increasing the number of dimensions of word vectors beyond 1000 actually reduces the performance of the model.

3) Visualization: In addition to quantitatively evaluating the word embeddings as described in the previous subsection, we can visualize them after their dimensionality is reduced with an algorithm like PCA (Principal Component Analysis). After such a step, usually one retains only a very small number of dimensions, typically 2 or 3.

A zoomed-in view for a few software-centric words when considering only the top two dimensions of PCA is shown in Figure 6. It can be readily observed from the figure that the words with similar meanings form distinguishable clusters in the 2-dimensional PCA space.

In addition to visualization, in Table V we show five interesting software terms in the first row along with their three most cosine similar terms in the following three rows.

To answer the questions that follow we will frequently refer to Tables VII and VIII. Table VII shows the evaluation results of the four different retrieval algorithms on two different datasets — Eclipse and AspectJ — under "title-only" setting, while Table VIII shows the same for "title+desc" queries. In the tables we show the percentage improvements and the p-values for each of the retrieval models vis-a-vis the FI BOW model along with the MAP values.

RQ2: Does the IR model that only incorporates ordering relationships improve the result over simple BoW IR model?

The IR model that only incorporates ordering relationships, i.e. MRF SD model, significantly beats the traditional FI BoW FI model in all four experiments — Eclipse title-only, AspectJ title-only, Eclipse title+desc, and AspectJ title+desc.

The improvement observed using MRF SD over FI BoW model ranges from 6% to 22% in different experiments with respect to MAP values. The $P@r$ and $R@r$ values are also significantly higher for MRF SD vis-a-vis FI BOW. We also observe that MRF SD model shows more improvement when run against Eclipse dataset as opposed to AspectJ dataset.

RQ3: Does the IR model that only incorporates semantic relationships improve the result over simple BoW IR models?

To answer the question posed above we again refer to the Tables VII and VIII. As we can observe from the tables the PWSM model that only incorporates term-term semantic relationships significantly outperforms the traditional FI BoW model with improvements ranging from 9% to 31% in terms of MAP values. The values for $P@r$ and $R@r$ are also significantly higher for PWSM in relation to BoW model.

The biggest improvement is observed in the retrieval experiment performed for the Aspectj software library using title-only queries. It can also be observed that the improvements for title+desc experiments are lower than the improvements for title-only experiments.

RQ4: How does our novel SCOR retrieval model perform against various BoW IR models?

This question requires comparing our SCOR retrieval model against various BoW based source code retrieval models — FI BOW, BugLocator [4], BLUiR [5], and SCP-QR [6].

Note that SCOR and FI BoW results are provided in Tables VII and VIII. When comparing SCOR against the simple FI BOW model we observe that the improvements observed in term of MAP values, by SCOR over FI BOW range from 17% for AspectJ title+desc queries to even 44% for AspectJ title-only queries.

For comparing SCOR with more advanced retrieval engines — BugLocator [4] and BLUiR [5] — we refer to the Figure 7. Note that for comparison we evaluate our SCOR results on the 3057 Eclipse and 286 AspectJ bug reports that were evaluated by the authors of BugLocator and BLUiR, and not on the entire 4035 Eclipse and 291 AspectJ bug report dataset whose evaluation results are given in the Tables VII and VIII. We observe from the chart provided in Figure 7, for Eclipse title+desc dataset our SCOR retrieval algorithm with MAP value of 0.343 beats BugLocator and BLUiR, which have MAP values of 0.300, and 0.320, respectively. Also, for AspectJ title+desc dataset, SCOR with MAP value of 0.255 outperforms BugLocator and BLUiR, which have MAP values 0.220 and 0.250, respectively. However, SCOR and BLUiR retrieval accuracies on AspectJ dataset are quite comparable.

Finally, we compare SCOR with SCP-QR [6]. The MAP value obtained using SCP-QR for 4035 Eclipse title-only

queries is 0.2296. In comparison, SCOR with MAP value of 0.2709 outperforms SCP-QR by 18%.

RQ5: How good is SCOR against pure MRF based retrieval techniques that only model ordering relationships?

The answer to this question requires comparing SCOR with the MRF based Sequential Dependence (SD) and Full Dependence (FD) source code retrieval models presented in [3]. Notice that the same MRF SD model used in [3] is an important component inside our SCOR retrieval framework.

As shown in the retrieval results provided in Figure 8 our SCOR retrieval model significantly outperforms both MRF SD and FD models on Eclipse and AspectJ title-only as well as title+desc queries. The improvements observed using SCOR over MRF SD range from 5.6% for Eclipse title+desc queries to 33% for AspectJ title-only queries, while the improvements observed using SCOR over MRF FD range from 5.6% for AspectJ title+desc to 27% for AspectJ title+desc queries. With regard to significance testing with the Student's Paired t-test [48], the p-values obtained when comparing SCOR with MRF SD model for different experiments are as follows: 3e-2 for Eclipse title+desc, 4e-3 for Eclipse tite-only, 3e-2 for AspectJ title+desc, and 3e-2 for AspectJ title-only.

RQ6: How does our SCOR retrieval model perform against other semantic embeddings based retrieval models?

To answer this question we compare our retrieval algorithm with two semantic embeddings based retrieval algorithms — the neural embeddings based retrieval algorithm proposed by Ye et al. [8], and Latent Semantic Analysis (LSA) based source code retrieval model proposed by Rao and Kak [7].

Note that while the model presented in [8] is composed of many relevance scores that linearly combine to produce a single composite score, we limit our focus only to the semantic portion of the model. Therefore, for the purpose of a comparative study, we only implemented the semantic embedding based scoring mechanism presented in [8] and linearly combined it with our baseline Dirichlet smoothed FI BoW model as provided in Equation (2). We compare this model against our PWSM retrieval model instead of the more powerful SCOR retrieval engine. We believe that such a comparison is fair in nature. Notice that we use the same word vectors we learn by applying the Skip-gram to 35000 repositories for both the algorithms. We notice that the MAP value obtained for PWSM for Eclipse title+desc is 0.2713, while the MAP value for Ye et al.'s algorithm is 0.2687.

With regards to comparing with LSA model proposed in [7], we notice that Rao and Kak used the same iBUGS AspectJ queries in their study as we have used in this paper. Therefore, a direct comparison is possible between PWSM and the models presented in [7]. The best MAP value reported by Rao and Kak for LSA on iBUGS using 291 queries is 0.0700, while the MAP value for PWSM on the same dataset with the same 291 queries is 0.2334.

RQ7: Are word2vec based word vectors generic enough to be used for searching in a new software library?

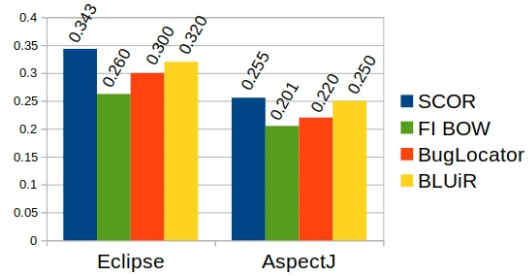


Fig. 7: Comparison of SCOR with various BoW models on Eclipse and AspectJ "title+desc" queries using MAP values.

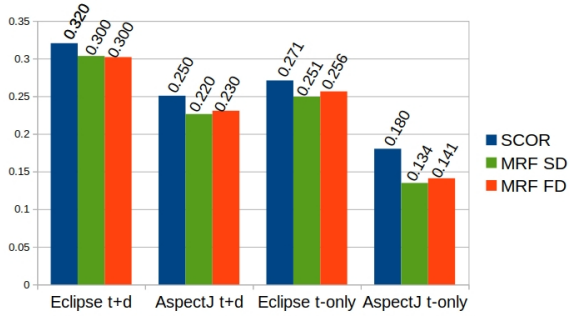


Fig. 8: Comparison of SCOR with MRF SD and FD models on Eclipse and AspectJ queries using MAP values.

The answer to this question is yes, because the AspectJ library on which we perform retrievals using iBUGS queries was not present in the training set when we generated the word embedding using the Skip-gram model. Yet, the improvements observed in retrieval precision when semantic embeddings based models are used for searching in the AspectJ library are very impressive as shown in Tables VII and VIII.

VI. Conclusion

We presented a novel source code retrieval framework that incorporates both ordering and semantic relationships between the terms, and show that it significantly improves the retrieval precision on two popular datasets —Eclipse and AspectJ. For the purpose of retrieving a file that is relevant to a query, our framework extends the basic word2vec neural architecture with two additional layers that we have called ML1 and ML2. While the ML1 layer is produced solely by comparing the terms in a query with the terms in a file by applying the Cosine distance measure to the numeric word embeddings produced by word2vec, the ML2 layer is produced by applying a 2×2 convolutional kernel to the ML1 layer. The convolutional kernel is designed such that the elements of the array of numbers in ML2 become high only when a file is sufficiently similar to a query both with respect to the terms and also with respect to the order in which the terms appear.

Our work reported here also involved running what is probably the largest experiment undertaken so far for creating the word embeddings for software-centric terms. We downloaded 35,000 Java repositories from GitHub, processed over half a million term involving about a billion tokens from these repositories to generate the numeric vectors for the terms.

REFERENCES

[1] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[2] D. Metzler and W. B. Croft, "A markov random field model for term dependencies," in *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2005, pp. 472–479.

[3] B. Sisman, S. A. Akbar, and A. C. Kak, "Exploiting spatial code proximity and order for improved source code retrieval for bug localization," *Journal of Software: Evolution and Process*, vol. 29, no. 1, 2017.

[4] 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 *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 14–24.

[5] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 345–355.

[6] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 309–318.

[7] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: a comparative study of generic and composite text models," in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 43–52.

[8] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 404–415.

[9] B. Sisman and A. C. Kak, "Incorporating version histories in information retrieval based bug localization," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 2012, pp. 50–59.

[10] L. Moreno, J. J. Treadway, A. Marcus, and W. Shen, "On the use of stack traces to improve text retrieval-based bug localization," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 151–160.

[11] C.-P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei, "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 181–190.

[12] S. Wang and D. Lo, "Version history, similar report, and structure: Putting them together for improved bug localization," in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 53–63.

[13] J. Peng, C. Macdonald, B. He, V. Plachouras, and I. Ounis, "Incorporating term dependency in the dfr framework," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 843–844.

[14] M. Bendersky, D. Metzler, and W. B. Croft, "Learning concept importance using a weighted dependence model," in *Proceedings of the third ACM international conference on Web search and data mining*. ACM, 2010, pp. 31–40.

[15] Y. Lv and C. Zhai, "Positional language models for information retrieval," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2009, pp. 299–306.

[16] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *11th Working Conference on Reverse Engineering*, Nov 2004, pp. 214–223.

[17] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003. [Online]. Available: http://www.jmlr.org/papers/v3/bengio03a.html

[18] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *AISTATS05*, 2005, pp. 246–252.

[19] A. Mnih and Y. W. Teh, "A fast and simple algorithm for training neural probabilistic language models," in *In Proceedings of the International Conference on Machine Learning*, 2012.

[20] B. Mitra, E. T. Nalisnick, N. Craswell, and R. Caruana, "A dual embedding space model for document ranking," *CoRR*, vol. abs/1602.01137, 2016. [Online]. Available: http://arxiv.org/abs/1602.01137

[21] G. Zuccon, B. Koopman, P. Bruza, and L. Azzopardi, "Integrating and evaluating neural word embeddings in information retrieval," in *Proceedings of the 20th Australasian Document Computing Symposium*, ser. ADCS '15. New York, NY, USA: ACM, 2015, pp. 12:1–12:8. [Online]. Available: http://doi.acm.org/10.1145/2838931.2838936

[22] D. Ganguly, D. Roy, M. Mitra, and G. J. Jones, "Word embedding based generalized language model for information retrieval," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15. New York, NY, USA: ACM, 2015, pp. 795–798. [Online]. Available: http://doi.acm.org/10.1145/2766462.2767780

[23] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, "Semantic matching by non-linear word transportation for information retrieval," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '16. New York, NY, USA: ACM, 2016, pp. 701–710. [Online]. Available: http://doi.acm.org/10.1145/2983323.2983768

[24] T. Kenter and M. de Rijke, "Short text similarity with word embeddings," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '15. New York, NY, USA: ACM, 2015, pp. 1411–1420. [Online]. Available: http://doi.acm.org/10.1145/2806416.2806475

[25] Y. Uneno, O. Mizuno, and E. Choi, "Using a distributed representation of words in localizing relevant files for bug reports," in *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Aug 2016, pp. 183–190.

[26] T. V. Nguyen, A. T. Nguyen, H. D. Phan, T. D. Nguyen, and T. N. Nguyen, "Combining word2vec with revised vector space model for better code retrieval," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, May 2017, pp. 183–185.

[27] S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra, "Retrieval on source code: A neural code search," in *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. MAPL 2018. New York, NY, USA: ACM, 2018, pp. 31–41. [Online]. Available: http://doi.acm.org/10.1145/3211346.3211353

[28] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, ser. CIKM '13. New York, NY, USA: ACM, 2013, pp. 2333–2338. [Online]. Available: http://doi.acm.org/10.1145/2505515.2505665

[29] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, ser. CIKM '14. New York, NY, USA: ACM, 2014, pp. 101–110. [Online]. Available: http://doi.acm.org/10.1145/2661829.2661935

[30] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 2042–2050. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969055

[31] Z. Lu and H. Li, "A deep architecture for matching short texts," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 1367–1375. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999611.2999764

[32] L. Pang, Y. Lan, J. Guo, J. Xu, S. Wan, and X. Cheng, "Text matching as image recognition," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 2793–2799. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016100.3016292

[33] B. Mitra, F. Diaz, and N. Craswell, "Learning to match using local and distributed representations of text for web search," in *WWW*, 2017.

[34] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information re-

trieval," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, May 2017, pp. 218–229.

[35] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 933–944. [Online]. Available: http://doi.acm.org/10.1145/3180155.3180167

[36] Y. Xiao, J. Keung, Q. Mi, and K. E. Bennin, "Improving bug localization with an enhanced convolutional neural network," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Dec 2017, pp. 338–347.

[37] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[38] "Scor word embeddings." [Online]. Available: https://engineering.purdue.edu/RVL/SCOR_WordEmbeddings

[39] X. Rong, "word2vec parameter learning explained," *CoRR*, vol. abs/1411.2738, 2014. [Online]. Available: http://arxiv.org/abs/1411.2738

[40] "Eclipse software." [Online]. Available: eclipse.org

[41] "Aspectj software." [Online]. Available: https://eclipse.org/aspectj

[42] "Buglinks dataset." [Online]. Available: https://engineering.purdue.edu/RVL/Database/BUGLinks/

[43] "ibugs dataset." [Online]. Available: https://www.st.cs.uni-saarland.de/ibugs/

[44] "Github website." [Online]. Available: https://github.com/

[45] X. Wang, L. Pollock, and K. Vijay-Shanker, "Developing a model of loop actions by mining loop characteristics from a large code corpus," 2015.

[46] "Gensim software." [Online]. Available: https://radimrehurek.com/gensim/

[47] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1, no. 1.

[48] A. C. Kak, "Evaluating information retrieval algorithms with significance testing based on randomization and students paired t-test." [Online]. Available: https://engineering.purdue.edu/kak/Tutorials/SignificanceTesting.pdf

[49] A. Bakarov, "A survey of word embeddings evaluation methods," *CoRR*, vol. abs/1801.09536, 2018. [Online]. Available: http://arxiv.org/abs/1801.09536

[50] Y. Choi, C. Yi-I Chiu, and D. Sontag, "Learning low-dimensional representations of medical concepts," *AMIA Joint Summits on Translational Science proceedings. AMIA Summit on Translational Science*, vol. 2016, pp. 41–50, 07 2016.