2017 24th Asia-Pacific Software Engineering Conference

An Improved Approach to Traceability Recovery Based on Word Embeddings

Teng Zhao
School of Computer Science and Engineering
Beihang University
Beijing, China
zhaoteng@buaa.edu.cn

Qinghua Cao
School of Computer Science and Engineering
Beihang University
Beijing, China
caoqinghua@buaa.edu.cn

Qing Sun*
School of Computer Science and Engineering, School of Economics and Management
Beihang University
Beijing, China
sunqing@buaa.edu.cn

***Abstract*—Software traceability recovery, which reconstructs links between software artifacts, has become more and more vital to maintaining a software life cycle with the increase of software scale and complexity of software architecture. However, existing approaches mainly rely on information retrieval (IR) techniques. These methods are not very efficient at complex software artifacts which are mixed with multilingual texts, code snippets and proper nouns. Moreover, it is hard to predict new traceability links with existing approaches when requirements are changed or software functions are added, since these methods have not made the most of the final ranked lists. In this paper, we propose a novel approach WELR, based on word embeddings and learning to rank to recover traceability links. We use word embeddings to calculate semantic similarities between software artifacts and bring in query expansion and a weighting strategy during calculation. Different from other work, we leverage learning to rank to build prediction models for traceability links. We conducted experiments on five public datasets and took account of traceability links among different kinds of software artifacts. The results show that our method outperforms the state-of-the-art method that works under the same conditions.**

Keywords—traceability recovery; word embeddings; learning to rank; semantic similarity

I. INTRODUCTION

Traceability recovery is used to discover relationships between thousands of software artifacts to facilitate the efficient retrieval of relevant information in large-scale industrial projects [1]. Complete and accurate traceability links can ensure each related elements will be considered when changing requirements and ensure every requirement is implemented, therefore traceability recovery play important roles in software maintenance [1], bug localizations [11, 35, 36] and etc. Traditional methods of recovering traceability include building requirement traceability matrices (RTMs), building requirement traceability graphs. However, these methods are difficult to extend and error-prone with the evolution of software [2]. Hence, many researchers put forward approaches to solve this problem with information retrieval (IR) techniques, and these methods are mainly based on text retrieval, e.g. VSM [3-5], LSA [3-5]. As highlighted in [2], text analysis techniques are used to solve more and more problems in software engineering.

While existing IR methods can recover traceability links, they are limited in several ways. First, the calculation of similarity between software artifacts is based on word frequency, regardless of semantic information. Second, they have the defects of bag-of-words (BOW), i.e. they ignore word orders in one sentence. Third, these methods bring the lexical gap problem because they suppose one word only has one meaning. As such, many semantic relationships in a sentence may be missed and result in loss of relevant information. Finally, computing large vectors that are represented by these methods is time-consuming and has low efficiency. In natural language processing (NLP), some approaches to calculate semantic similarities between texts, e.g. Word2vec [6, 9] and Doc2vec [10], are proposed. Although they overcome the shortcomings existing in IR methods, it turns out that they are not very suited for the tasks which involve complicate types of text files and files that consist of multilingual texts, like software artifacts. Therefore, calculating sematic similarities provides more accurate results than the methods based on word frequency when recovering traceability links and besides, documents should be preprocessed in terms of the specificity of software artifacts.

Learning to rank (LtR) refers to machine learning techniques for training the model in a ranking task [31]. It has been frequently used in many applications in IR, NLP and data mining. However, to the best of our knowledge, few work use LtR to further handle the ranked lists after IR process in the field of traceability recovery. LtR can make use of the information, which is probably omitted by IR process, to

* Correspondence should be addressed to Qing Sun; sunqing@buaa.edu.cn.

provide a supplement to the task of traceability recovery. Thus, the use of LtR in this task can enhance the performance and improve the precision.

In this paper, we propose a novel approach, called WELR, which is based on word embeddings and learning to rank to recover traceability links. First, we combine query expansion and a weighting strategy with word embeddings to build primary traceability links. Second, we apply learning to rank technique to improve the precision of results generated from the first step.

Compared with previous approaches to this task, we have made three contributions. First, WELR provides an improved method to calculate semantic similarities between software artifacts by using word embeddings and adopting query expansion and weighting strategy. It is much more effective and more generic to recover traceability links than IR methods. Second, WELR integrates learning to rank technique with similarity computation between software artifacts. It improves the precision of retrieval results. Third, WELR provides prediction models for traceability links, and thus makes traceability recovery extensible.

The rest of this paper is organized as follows. Section II describes related applications of text similarity and learning to rank in software engineering. In Section III, we present our traceability recovery approach. And Section IV details the experiments and results analysis, followed by conclusions and future work in Section V.

II. Related Work

In this section we discuss previous work and methods related to different aspects of our work.

A. *Text Retrieval in Software Engineering*

More and more software engineering (SE) tasks are addressed with text retrieval (TR) techniques [12], for example , traceability recovery [4, 5], feature location [7], software reuse [7] and etc. Many approaches have been proposed to improve the retrieval performance, and some of them related to our work are listed as follows.

The first one is automatic query expansion (AQE). AQE has been widely used in information retrieval tasks. It extends the query terms using words that share statistical relationships or meanings with query terms [22], and provides a solution for "the vocabulary problem" [14], which is the mismatch between terms in queries and vocabularies in corpus. C. Carpineto [13] points that most promising AQE methods explicitly take term dependency into account, e.g., combination of statistical and linguistic techniques. Besides, there are also many strategies about AQE, for example, [15] introduces a method for expanding query with re-weighting query terms, which can expand query with synonym terms, together with hyponyms and hypernyms.

The second one is word embeddings, also known as word vectors. Word embeddings have recently attracted many researchers on word representation and document representation. There are two architectures for training word embeddings: skip-gram and continuous bag-of-words (CBOW) [8, 9]. And recently several works propose improved theory model for training word embedding, like [40]. [40] proposed a probabilistic language model by combining word2vec with a latent continuous time series. Many text retrieval tasks in SE have leveraged word2vec or an improvement of word2vec. In [11], a similar work to us, the authors propose a method of learning vectors on software documents and using word embeddings to calculate document similarities. In that paper, two different training settings, one-vocabulary setting and two-vocabulary setting, are proposed with consideration of the existence of polysemy. One-vocabulary setting mixes natural language and artifacts token or source code token together, while two-vocabulary setting splits them up. However, since they have no consideration of some software documents in which natural language texts and source code are mixed, their method is not generic for all software artifacts.

Jin G. et al. [6] introduce another improvement of word2vec technique in which word embeddings are combined with recurrent neural networks (RNN) to recover traceability links, and the semantic likelihood of two artifacts is predicted by RNN. Their method performs well on large industrial software datasets which include artifacts and traceability links. However, to some relatively small software datasets, the performance is not as good as expected. Moreover, in fact, it is hard to get such large industrial datasets which contains sufficient traceability links for training RNN model.

Another work in [3] is similar to [6] to some extent, and [3] provides a method using ML classifiers to estimate the number of traceability links. Importantly, in [3], the authors highlight the primary reason of using classifier, rather than just setting a threshold on determining traceability links, is that an NLP technique preforms differently on different thresholds, therefore for each type of traceability links or each software dataset, a given threshold may not be suitable. Hence, we are motivated to combine a machine learning technique with text retrieval method in our work.

Tien-Duy B. Le et al. [35] integrate IR and program spectrum for bug localization, and they use vector space model (VSM), Tarantula and a method based on suspicious words to calculate three different similarities between different artifacts respectively and finally integrate them together. And Shaowei Wang et al. [36] propose a similar approach to locate bugs. They use five kinds of sources to locate relevant buggy files and finally integrate five results together. Bug localization is similar with traceability recovery tasks, and in Section IV, we conducted experiments to compare our approach against theirs. In [37] the authors propose a multi-abstraction concern localization technique, which combines VSM and multiple topic models (aka. LDA model). The points for their work is that they use generic algorithm to estimate a near-optimal configuration of the topic models. While their methods performs well on specific datasets, they also have the defect of bag-of-words.

Another related work is [41], which is used for discovering relevant tutorial fragment for APIs. The authors present an unsupervised approach leveraging topic model and PageRank algorithm. Their method overcomes the defect of supervised approaches, like pre-required annotated data and highly

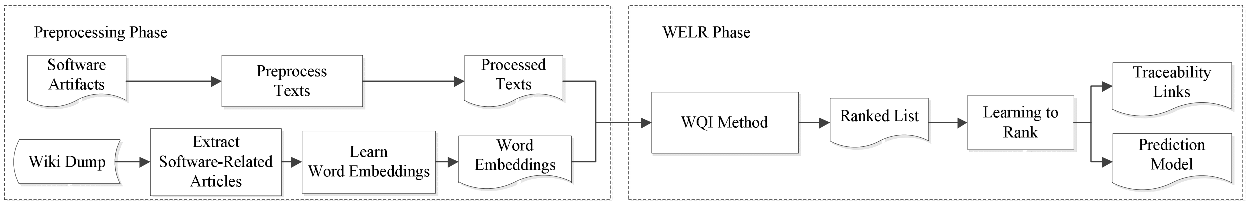


Fig. 1. The process of our proposed approach to traceability recovery

dependency of the characteristics of corpora. Their approach leverages many characteristics of fragments, hence it is hard to be applied to generic tasks of traceability recovery.

In addition, software artifacts, different from common articles, include requirement documents, design documents, source code and etc. Most of them are consisted of multilingual texts, code snippets and proper nouns. It leads to differences between software artifacts and articles written in natural language, thus common text retrieval methods are not suitable for this task.

B. Learning to Rank in Software Engineering

As a supervised or semi-supervised ML method [16], learning to rank plays an important role in information retrieval, data mining and NLP tasks though building ranking models. And in software engineering, LtR is often used for bug localization [20, 38, 39], fault location [28] and duplication detection [29], as it provides a valid and convenient way to make use of software artifacts' features that, for example, exploit the levels of different documents and the semi-structure nature of source code [18]. For each query and candidate documents pair, like $(query,\ candidate\ documents)$, we can extract several features (e.g. lexical similarity, semantic similarity) and use them as the input of ML models. More formally, let q_i is ith query and $D_i = \{d_{i_1}, d_{i_2} \cdots d_{i_j} \cdots d_{i_n}\}$ is corresponding candidate documents set to q_i, the ranking function is as follows:

$$R = h(w, \varphi(q_i, D_i)) \qquad (1)$$

where R is the final ranking result, w is a weight matrix in which each value stands for the weight of feature, function $\varphi(\cdot)$ maps query-document pairs to feature vector representation and function $h(\cdot)$ is the ranking approach. There are three options for function $h(\cdot)$: pointwise, pairwise and listwise. Pointwise approach only considers the absolute correlation about a single document with a specific query, while pairwise considers correlations between each two documents. Although pairwise approach has an improvement than pointwise approach, it takes the cost of considering numerous pairs of documents. Listwise approach considers the whole sequence of documents list. On the one hand listwise approach can produce the best results among the three approaches, on the other hand it is also time-consuming and memory-consuming since it always generates the complete order of candidate links especially when the dataset is large.

Appropriate selection of features for LtR can make great differences on the trained model of LtR approach [21]. Almost work using LtR selects similarities between words or texts as one feature in software engineering tasks. In [20], the authors select surface lexical similarity, API-enriched lexical similarity and class name similarity, and in [21], text similarity and context similarity are selected as features. It should be noted that they all use cosine similarity when calculating similarities. Besides, the authors in [21] divide all features into two groups: query-dependent features and query-independent features. Query-dependent features are related with queries, like similarity, while query-independent features only reflect characteristics of candidate results regardless of the query, like terms frequencies [21], bug-fixing frequency [20] and etc. The number of selected features should not be too small in case of inaccuracy and also should not be too large in case of over-fitting problem. In our work, we are inspired by [21] to select query-dependent and query-independent features.

As for ranking approaches, in software engineering [19] selects Bayesian logistic regression and linear regression as training algorithm. Differently, RankBoost, a pairwise approach, is used in [21] which builds links between queries and code, and results' rank are sensitive. In the task of traceability recovery, we are only interest in the relative order of two candidate traceability links, rather than the complete order of a ranked list. Thus we select a pairwise approach. In our work, we select the pairwise approach Ranking SVM that has been proved to be efficient in [11], and details will be introduced in Section III.

III. METHOD

In this section, the overall process is listed, which is followed by the details of key steps in our method, including query expansion and weighting strategy, semantic similarity and learning to rank.

A. Method Overview

The process of our traceability recovery method, as depicted in Fig. 1, is composed of preprocessing phase and WELR phase. Preprocessing phase consists of text preprocessing and word embeddings learning. And WELR phase consists of two main steps of our work: 1) using WQI method, which is composed with word embeddings, query expansion and the weighting strategy IDF, to get ranked list; 2) determining traceability links and building a prediction model.

Preprocessing phase will be detailed in Section IV, and in the next subsections, we will introduce the details of WQI method and learning to rank.

B. Query Expansion and weighting strategy

In this paper, semantic similarity between two words will be calculated by cosine similarity (2) using word embeddings.

$$sim_{w2w}\left(w_i, w_j\right) = \cos\left(\boldsymbol{w_i}, \boldsymbol{w_j}\right) = \frac{\boldsymbol{w_i}}{\left\|\boldsymbol{w_i}\right\|} \cdot \frac{\boldsymbol{w_j}}{\left\|\boldsymbol{w_j}\right\|} \quad (2)$$

where $\boldsymbol{w_i}$ and $\boldsymbol{w_j}$ are the vectors of w_i and w_j respectively and are represented with word embeddings. $\left\|\boldsymbol{w_i}\right\|$ and $\left\|\boldsymbol{w_j}\right\|$ are the magnitude of vectors.

The use of appropriate query expansion (QE) approaches can enhance the performance of IR tasks [22], and so is the weighting strategy. We select inverse document frequency (IDF) as the weighting strategy which can indicates the importance of a term in the scope of all documents and it has been proved to be efficient [32]. There are two reasons for selecting IDF rather than TF or TF-IDF. The one reason is that we pay more attention on the common important words in each software dataset. And another reason is that we can get all terms' IDF values by traversing all documents at only one time, which can reduce process time. The process of using QE and IDF is as follows. First, sort terms in processed text S by IDF weights which are calculated in advance, and then select terms whose weight scores are at the top $topn\%$. Selected terms form an expansion word set EXT_S of S together. Next, expand the selected terms in EXT_S with similar words. The two steps above can be formally represented as (3).

$$\begin{aligned} &QE\_SET_w = \left\{word \middle| sim_{w2w}\left(w, word\right) > \delta_{qe}\right\} \\ &\text{s.t. } w \in EXT_S \end{aligned} \quad (3)$$

where QE_SET_w is the expansion set of word w , δ_{qe} is the similarity threshold which is used to limit the number of similar words in order to reduce noisy words. In our work, $topn\%$ and δ_{qe} are set empirical value 0.3 and 0.7 respectively.

C. Semantic Similarity

In [11], the similarity of a word w and a bag of words T is calculated as the maximum similarity between w and any word w' in T :

$$sim_{w2t\_ori}(w,T) = \max_{w' \in T} sim_{w2w}\left(w, w'\right) \quad (4)$$

And the similarity between T_i and T_j is calculated by the following equation:

$$sim_{t2t\_asy}\left(T_i \to T_j\right) = \frac{\sum_{w \in P\left(T_i \to T_j\right)} sim_{w2t\_ori}(w, T_j)}{\left|P\left(T_i \to T_j\right)\right|} \quad (5)$$

where $P\left(T_i \to T_j\right)$ is the set of words in T_i that have positive similarity with T_j. In this paper, we call the method in [11] as W2V method. This equation is asymmetric hence different order of T_i and T_j would work out different values.

In our model, we rewrite (4) by adding a query expansion item and assigning proportions of each item in equation with parameter α .

$$sim_{w2t}\left(w,T\right) = \max_{w' \in T}\left\{\alpha \cdot sim_{w2w}\left(w,w'\right) + \left(1-\alpha\right) \cdot g\left(w,w'\right)\right\} \quad (6)$$

subject to:

$$g(w,w') = \frac{\sum_{w_k \in QE\_SET_w} \lambda_k \cdot sim_{w2w}(w_k, w')}{\left|QE\_SET_w\right|} \quad (7)$$

$$\lambda_k = sim_{w2w}(w, w_k) \quad (8)$$

where function $g(w,w')$ is the query expansion item. In (7), λ_k represents the similarity between w and a word w_k in QE_SET_w , and $\left|QE\_SET_w\right|$, which represents the size of expanded list, is used for normalization to ensure function $g(w,w')$ varies from -1 to 1. When the word w is in EXT_S , i.e., the value of IDF of the word w is at the top $topn\%$ in text S , the calculation of the similarity between a word in S and T should add query expansion item and use the parameter α to adjust the proportion of each part. The parameter α is tuned in the empirical range from 0.5 to 0.9 with the step of 0.01, and finally α is set by the optimal value which leads the best result. Therefor the parameter α is different in different datasets. And if the word w is not in EXT_S , we use the same method with [11], i.e. the equation (4), to calculate the similarity between a word and a text.

After defining the method of calculating similarity between a word and a text, we can get similarity between two texts by substituting $sim_{w2t\_ori}(w,T)$ with $sim_{w2t}\left(w,T\right)$ in (5). Finally, we set a threshold value *finalThreshold* to filter the ranked list, and only the artifacts whose similarity with the query artifact beyond this value would be selected. The threshold value *finalThreshold* is tuned in an empirical range, and this parameter is also different in different datasets.

TABLE I. SELECTED FEATURES

Features	Description	Query-dependent
Semantic Similarity	Similarity between artifacts with our proposed method	yes
Generalized Jaccard Coefficient	Similarity between artifacts which are represented by the average of word vectors in texts	yes
Sum of IDF	Sum of IDF of query terms in candidate results	yes
Number of Keywords	Number of words which is at the top **topn**% of sorted terms by IDF scores in a candidate result	no
Line Length	Length of a candidate result	no

Moreover, a restriction that T_i is longer than T_j is set. There are two reasons for this restriction: 1) making the equation symmetric; 2) avoiding some terms in the longer text being overlooked in the case of that the shorter one is a subset of the longer one [23].

D. Learning to Rank

This section details feature selection and ranking approach of learning to rank technique.

1) Feature Selection

We take five features as the input of LtR model based on the characteristics of our proposed text similarity method. And like [21], five features are divided into two groups: query-dependent features and query-independent features (TABLE I.).

Semantic Similarity is calculated by the method presented in Section III.B. *Generalized Jaccard Coefficient* shows the context similarity about sentence contexts of a query and a candidate artifact, which is calculated by (10).

$$EJ\left(\boldsymbol{x},\boldsymbol{y}\right)=\frac{\boldsymbol{x}\cdot\boldsymbol{y}}{\left\|\boldsymbol{x}\right\|^2+\left\|\boldsymbol{y}\right\|^2-\boldsymbol{x}\cdot\boldsymbol{y}} \tag{9}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are document vectors, which are represented by the average of word embeddings of respective text.

The third feature *Sum of IDF* is used to remark the importance of query terms in a candidate result, by adding the IDF of query terms that occur in candidate text together.

Number of Keywords and *Line Length* both represent the effectiveness of a candidate text, by counting keywords of the candidate text and calculating the length of the candidate text respectively.

As shown in TABLE I. , the first three features are query-dependent, hence they emphasize the relationships between queries and candidate results. On the contrary, the last two features are query-independent, which represent more characteristics of candidate results.

2) Ranking Approach

We select Ranking SVM as the ranking approach as we mentioned in Section II. Performing steps are as follows. First, Ranking SVM maps query-document pairs to feature space [26]. Second, Ranking SVM calculates the distance between any two feature vectors. Finally, Ranking SVM transfers the ranking problem to classification problem, and solves this problem with regular SVM solver [26]. The optimization problem is formalized as follows.

$$minimize: \tfrac{1}{2}\left\|\mathbf{w}\right\|^2 + C\sum_{i=0}^{m}\xi_i \tag{10}$$

subject to:

$$y_i\cdot\boldsymbol{w}^T(x_i^{(1)}-x_i^{(2)})\geq 1-\xi_i \tag{11}$$

$$\xi_i \geq 0, i=1,\ldots,m \tag{12}$$

where $\boldsymbol{w}$ is a parameter vector, x is a feature vector, y represents relative correlation between a pair of documents and ξ is used as a slack variable. This function is to reduce the number of discordant query-result pairs as much as possible. In this paper, we use a practical tool, SVM^{rank} package, which has integrated the Ranking SVM algorithm.

IV. Experiments

This section first introduces our selected datasets and experiment settings, and next details text preprocessing phase, then proposes the baseline method, and finally shows the results and analyzes the results from several different aspects.

A. Dataset Description and Experiment Settings

In this paper, we choose five experiment datasets that are often used in the task of traceability recovery from CoEST website [1] : CM1-NASA, GANTT, eTOUR, iTrust and EasyClinic. CM1-NASA is a subset of a science instrument called CM1 developed by NASA, and eTOUR and EasyClinic both are java projects used for tour guide and hospital management, respectively. GANTT is a tool for managing project schedule by Gantt charts. And iTrust is a medical records web system in java. More detail messages about datasets are described in TABLE II. . CoEST provides not only test data but also traceability links among artifacts. The authoritative links can help validate our experiment results, and moreover, eliminate the labeling operations in the period of learning to rank.

And next we generate training corpus for word embeddings. Siwei L. [24] notes that when training word embeddings, the corpus domain is more important than the corpus size, and using an in-domain corpus significantly improves the performance for a given task. And for corpus in the same domain, a larger corpus will be better. Thus, to ensure the domain correlation, we download a software terms list, get rid of one-word terms, and attach those processed terms with specific word "software" to filter articles in wiki dumps[2]. After above steps, we get a 264M article set which contains 43,443,648 words. Finally, we concatenate experiment data with the article set for the final training corpus.

To learn word embeddings, we make use of a topic modeling tool genism [33] on our generated training corpus. And we choose CBOW model, set window size to 5 and set dimensions of word vector to 200.

TABLE II. EXPERIMENT DATASETS

Dataset Name	**Specification**	**Number of Tokens**
CM1-NASA	22 high-level requirements, 53 low-level requirements, 45 correct links	5767
GANTT	17 high-level requirements, 69 low-level requirements, 68 correct links	2080
eTOUR	58 user cases, 116 code classes, 308 correct links	97452
ITrust	131 user cases, 367 code classes, 534 correct links	123501
EasyClinic	30 user cases, 20 interaction diagrams, 63 test cases, 47 class descriptions, 1257 correct links	21882

[1] http://www.coest.org
[2] https://dumps.wikimedia.org/

B. *Text Preprocessing*

In this experiment documents are divided into two types: 1) text files, including: high-level and low-level requirements, user cases, interaction diagrams, test cases, class description, etc. 2) source code files.

First, we perform the same tokenization operation on text files and source code files: we split texts into bag of words by whitespaces, then remove punctuations, numbers and stop-words. In the period of removing stop-words, specific to text files, we make use of the stop words list provided by nltk [34] and append some useless words in specific dataset and specific to source code files, we treat java and C language keywords as stop-words.

Second, we split compound words by upper camel case rule. What calls for special attention is that only words that have no appearance in software terms list can be split, i.e., only the compound words defined by developers or users can be split. For example, "DefaultMutableTreeNode" will be split to four partitions: "Default", "Mutable", "Tree" and "Node". Then we convert all words to lower case. Note that there is no need to stem words or lemmatization.

Finally, we collect the files belonging to the same category to one file in order to make the next operations more convenient, for example, all user case files are grouped to a new file called “UC.txt”. And in the new file each original file takes up only one line.

C. *Baseline Method*

As mentioned in Section III, WELR phase is divided into two steps: 1) getting ranked list with WQI method; 2) performing learning to rank. Thus for the first step, we use LSI [30] and the method of calculating text similarity with (5), which is mentioned in Section III and is called W2V method, as baseline methods to compare WQI method. In this paper LSI specifically refers to using latent semantic indexing technique to recover traceability links and it is proved to be a mature method for this task [7]. As discussed in Section II, both LSI and W2V are natural baselines to WQI method. And for the second step, we compare some metrics on the same datasets before and after performing LtR. Moreover, we also conducted an experiment to compare our approach WELR against the state-of-the-art method AML which is proposed in [35].

TABLE III. RESULTS FOR THREE METHODS

Datasets		LSI		W2V		WQI	
		PRE	REC	PRE	REC	PRE	REC
CM1-NASA	HL→LL	0.127	0.41	0.262	0.217	0.371	0.329
GANTT	HL→LL	0.286	0.332	0.278	0.418	0.255	0.563
eTOUR	UC→CC	0.077	0.221	0.098	0.332	0.088	0.415
iTrust	UC→CC	0.009	0.45	0.192	0.363	0.198	0.322
Easy Clinic	UC->ID	0.259	0.833	0.338	0.75	0.342	0.806
	UC->TC	0.45	0.755	0.522	0.867	0.499	0.867
	UC->CC	0.317	0.503	0.215	0.677	0.232	0.76

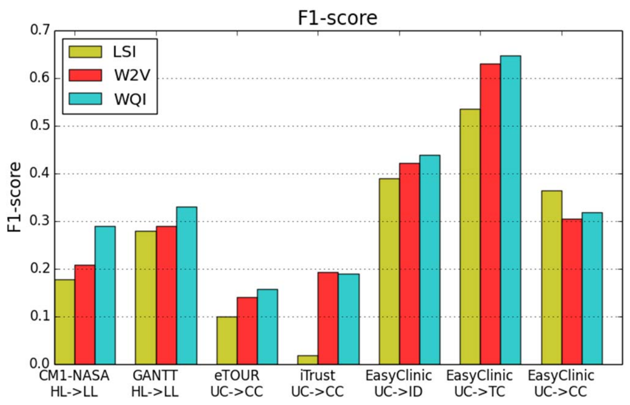


Fig. 2. F1-score of LSI, W2V and WQI on seven groups of data

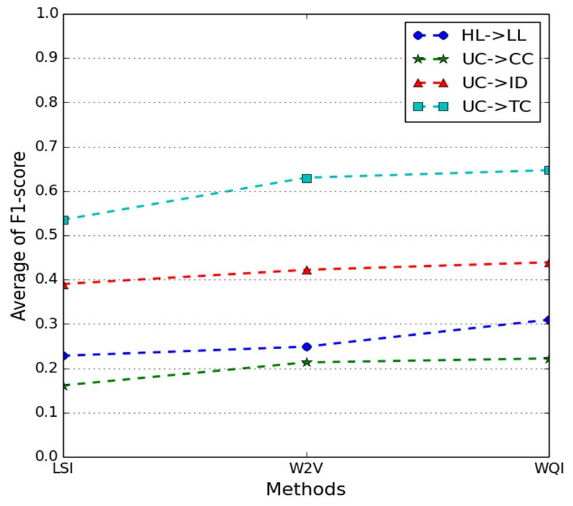


Fig. 3. Average F1-score of LSI, W2V and WQI method on different link types

D. *Results and Analysis*

In this section we show the results of our experiments. The method mentioned in Section III has been implemented and several common metrics in IR are used for evaluation, like Precision, Recall, F1-score, MRR, MAP and etc.

1) *Step1: Using WQI method*

In order to get the ranked list, we conducted three experiments using LSI, W2V and WQI method respectively. Every experiment works on seven groups of data.

As the results shown in TABLE III. , in which PRE stands for precision and REC stands for recall, the last two methods using word embeddings perform better than LSI method in average. Specifically, when using WQI, there will be a 33.3% relative improvement than LSI and a 6.6% relative improvement than W2V in precision, and a 24.5% relative improvement than LSI and a 17.1% relative improvement than W2V in recall. And Fig. 2 depicts the average F1-score on seven groups of data with three methods, and it shows that

WQI performs better than either LSI or W2V on almost datasets.

An important observation is that WQI has a general improvement for all types of traceability links listed in TABLE III. . The types of traceability links can be separated into two groups according to document types: text file to text file (t2t) and text file to source code file (t2c). In Fig. 3, HL, LL, UC, CC, ID and TC represent high-level requirement, low-level requirement, user case, class code, interaction diagram and test case respectively. And the traceability links HL to LL, UC to ID and UC to TC belong to t2t and UC to CC belongs to t2c. As shown in Fig. 3, WQI method improves the average F1-score for each type of traceability links. As a result, the improvement on t2t type of links proves that our semantic similarity calculation method is applicable to the task of traceability recovery. Furthermore, the improvement on t2c type of links shows that our preprocessing operations can effectively represent source code file as similarly as short text.

Compared with W2V, WQI brings in query expansion and IDF and rewrites the calculation methods. From the results, the changes improve the precision of the results. Query expansion extends queries with terms which are similar to corresponding query, and these terms occurs frequently in the relevant documents [25]. From another perspective, every word embedding contains semantic relationships among words and the feature of frequency, therefore the group of similar words contains more information than a single word. Besides, IDF, as a weighting strategy, not only reduces semantic noises but also reduces running time by abstracting keywords instead of using all words in a text for expansion.

To summarize, our proposed WQI method, which is used to calculate semantic similarities between software artifacts, can yield better performance than LSI and W2V on generating primary ranked lists.

2) *Step2:Using learning to rank*

Learning to rank is used to improve results by taking advantage of features of queries and ranked lists. In order to validate whether learning to rank technique benefits the results of traceability recovery, we conducted an experiment to compare the results before and after performing LtR. In our work, Ranking SVM are adopted as LtR method. And two common metrics are used in this experiment: Mean Reciprocal Rank (MRR), which is the multiplicative inverse of the rank of the first correct answer in candidate results, and Mean Average Precision (MAP), which is the mean of the average precision values for all queries. We select a mature pairwise model Ranking SVM and five features (mentioned in Section III) to train a ranking model. We conduct a ten-fold cross-validation, and average the results generated in each fold as the final result.

Fig. 4 depicts MRR and MAP score before and after LtR. Compared with the result before LtR, LtR method yields better performance in average. And specifically after LtR, MRR increases by 3.5% and MAP increases by 15.9% relatively. The significant increase on MAP shows that SVM^{rank} improves the overall ranking level and average retrieval precision. In other words, after the process of LtR, our model provides more accurate traceability links.

Moreover, after LtR, ranking models for each software dataset are generated. Ranking models contain almost features of software datasets, and many semantic relationships of software artifacts which are calculated by WQI method. Therefore, ranking models are regarded as prediction models, when requirements are changed or software functions are added, on condition of the changes of software are relatively small.

3) *Compare WELR against AML*

AML is proposed by Tien-Duy B. Le et al. in [35], which combines IR and program spectrum for bug localization and integrates three types of similarities together for final result. AML is a novel multi-modal bug localization approach and performs well on several software projects, including AspectJ, Ant, Lucene and Rhino. Moreover, it is generic and efficient in bug localization. Since the similarity of bug localization and our works, we choose this method as one of the baseline methods. We conducted this experiment on three datasets eTOUR, iTrust and EasyClinic, all of which contain source code files. As proposed method in [35], we calculated AML^{Text}, $\text{AML}^{Spectra}$ and $\text{AML}^{SuspWord}$ in turn, and finally integrate them as (13).

$$f(x_i,\theta) = \alpha \times AML^{Text} + \beta \times AML^{Spectra} + \gamma \times AML^{SuspWord} \quad (13)$$

where x_i represents a specific combination of user case, program spectrum and method name in code in our datasets and θ is the parameter vector $[\alpha, \beta, \gamma]$, which will be tuned by the probabilistic learning approach proposed in [35]. We also use the metrics MRR and MAP for evaluation.

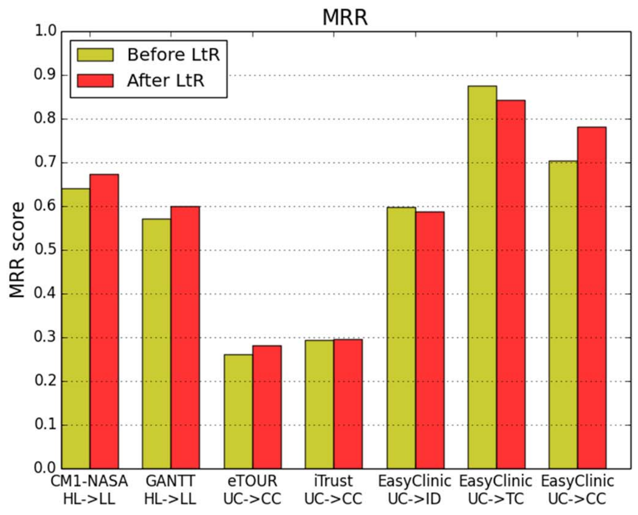
TABLE IV. and TABLE V. show the performance of AML and WELR in terms of MRR and MAP respectively. WELR improves the MRR score and the MAP score by 30.8% and 11.8% respectively than AML. Although in [35] the authors point that their AML method performs better than Ranking SVM which is also used in WELR, we can get more accurate results by combining word embeddings and Ranking SVM.

TABLE IV. COMPARISON OF MRR BETWEEN AML AND WELR

Datasets	**AML**	**WELR**
eTOUR	0.196	0.281
iTrust	0.247	0.296
EasyClinic	0.605	0.781

TABLE V. COMPARISON OF MAP BETWEEN AML AND WELR

Datasets	**AML**	**WELR**
eTOUR	0.134	0.153
iTrust	0.122	0.137
EasyClinic	0.460	0.501



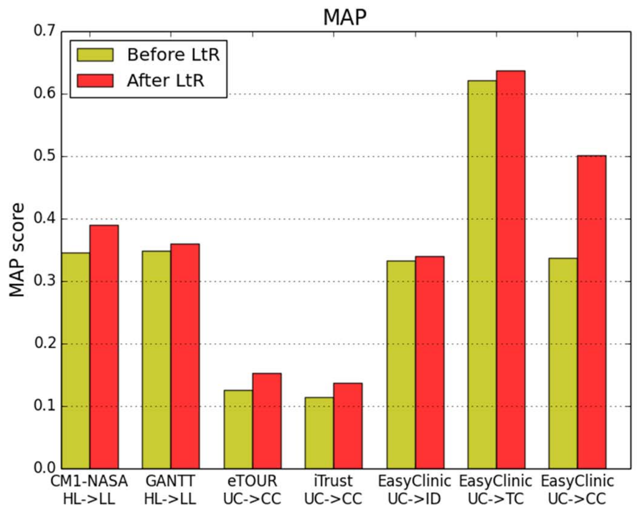


Fig. 4. Comparison of MRR and MAP before and after LtR

4) *Running Time*

TABLE VI. shows means of WELR's running time for different datasets and each dataset's information. Mean R.T.(sec) refers to the mean running time for recovering one traceability link on a specific dataset. The unit of running time is second. Retrieval Nums refers to the number of documents which need to be retrieved for each query, and Mean Documents' Length represents the mean documents' length in each dataset. According to the results, we can infer that the mean running time is positive correlated with retrieval numbers and mean documents' length. Our program runs on a Linux server with 8 i7 CPUs and 16G memory.

V. CONCLUSIONS AND FUTURE WORK

This paper proposes a generic and flexible approach to software traceability recovery. First, we propose an improved approach, called WQI, to calculate semantic similarities with word embeddings. In WQI, query expansion and a weighting strategy IDF are used to improve retrieval precision and efficiency. Second, we use Ranking SVM, an effective learning to rank algorithm, to rerank primary ranked list generated in first step and finally get traceability links and prediction models for each software dataset. We have experimented on several public software datasets which contain different kinds of software artifacts, then compared our method and results with LSI and a previous method which is also based on word embeddings. Moreover, through comparing our method WELR with one state-of-the-art method AML[35], WELR is more accurate on the task of traceability recovery. Our study shows that our approach is efficient at recovering traceability links, and effectively transfers word-level similarity to text-level similarity within the scope of the software engineering. We also demonstrate that our model outperforms the state-of-the-art method that works under the same conditions.

In our work, syntactic structures of complete sentences are broken because of eliminating stop words in the phase of text preprocessing. In the future, we plan to explore a method which can make use of syntactic structure information on the basis of the approach proposed in this paper. And in order to get more accurate traceability links, we plan to explore more effective features of queries and candidate results for the learning to rank algorithm.

TABLE VI. RUNNING TIME OF WELR AND RELATED INFORMATION

Datasets		**Mean R.T.(sec)**	**Retrieval Nums**	**Mean Documents' Length**
CM1-NASA	HL→LL	2.44	53	339
GANTT	HL→LL	2.05	69	107
eTOUR	UC→CC	49.42	116	1766
iTrust	UC→CC	79.25	367	2867
Easy Clinic	UC->ID	18.09	20	604
	UC->TC	27.00	63	604
	UC->CC	29.78	30	604

REFERENCES

[1] O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," Proceedings of IEEE International Conference on Requirements Engineering, Colorado Springs, CO, 1994, pp. 94-101.

[2] D. Port, A. Nikora, J. H. Hayes and L. Huang, "Text Mining Support for Software Requirements: Traceability Assurance," 2011 44th Hawaii International Conference on System Sciences, Kauai, HI, 2011, pp. 1-11.

[3] D. Falessi, G. Cantone and G. Canfora, "Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques," in IEEE Transactions on Software Engineering, vol. 39, no. 1, pp. 18-44, Jan. 2013.

[4] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella and S. Panichella, "Improving IR-based Traceability Recovery Using Smoothing Filters," 2011 IEEE 19th International Conference on Program Comprehension, Kingston, ON, 2011, pp. 21-30.

[5] Capobianco, Giovanni, et al. "Improving IR-based traceability recovery via noun‐based indexing of software artifacts." Jornal of Software: Evolution and Process 25.7 (2013): 743-762.

[6] Guo, Jin, Jinghui Cheng, and Jane Cleland-Huang. "Semantically enhanced software traceability using deep learning techniques." Proceedings of the 39th International Conference on Software Engineering. IEEE Press, 2017.

[7] Cleland-Huang, Jane, et al. "Software traceability: trends and future directions." Proceedings of the on Future of Software Engineering. ACM, 2014.

[8] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[9] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

[10] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." Proceedings of the 31st International Conference on Machine Learning (ICML-14). 2014.

[11] X. Ye, H. Shen, X. Ma, R. Bunescu and C. Liu, "From Word Embeddings to Document Similarities for Improved Information Retrieval in Software Engineering," 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, 2016, pp. 404-415.

[12] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 842-851.

[13] Carpineto, Claudio, and Giovanni Romano. "A survey of automatic query expansion in information retrieval." ACM Computing Surveys (CSUR) 44.1 (2012): 1.

[14] Furnas, George W., et al. "The vocabulary problem in human-system communication." Communications of the ACM 30.11 (1987): 964-971.

[15] E. Pyshkin and V. Klyuev, "A study of measures for document relatedness evaluation," 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), Wroclaw, 2012, pp. 249-256.

[16] X. Dong, X. Chen, Y. Guan, Z. Yu and S. Li, "An Overview of Learning to Rank for Information Retrieval," *2009 WRI World Congress on Computer Science and Information Engineering*, Los Angeles, CA, 2009, pp. 600-606.

[17] Severyn, Aliaksei, and Alessandro Moschitti. "Learning to rank short text pairs with convolutional deep neural networks." Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 2015.

[18] D. Binkley and D. Lawrie, "Learning to Rank Improves IR in SE," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, 2014, pp. 441-445.

[19] Y. Zou, T. Ye, Y. Lu, J. Mylopoulos and L. Zhang, "Learning to Rank for Question-Oriented Software Text Retrieval (T)," 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, 2015, pp. 1-11.

[20] Ye, Xin, Razvan Bunescu, and Chang Liu. "Learning to rank relevant files for bug reports using domain knowledge." Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014.

[21] Niu, Haoran, Iman Keivanloo, and Ying Zou. "Learning to rank code examples for code search engines." Empirical Software Engineering 22.1 (2017): 259-291.

[22] Alnofaie, Sara, Mohammed Dahab, and Mahmoud Kamal. "A Novel Information Retrieval Approach using Query Expansion and Spectral-based." information retrieval 7.9 (2016).

[23] Kenter, Tom, and Maarten De Rijke. "Short text similarity with word embeddings." Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, 2015.

[24] S. Lai, K. Liu, S. He, J. Zhao, "How to Generate a Good Word Embedding?," in IEEE Intelligent Systems , vol.PP, no.99, pp.1-1

[25] Roy, Dwaipayan, et al. "Using word embeddings for automatic query expansion." arXiv preprint arXiv:1606.07608 (2016).

[26] Joachims, Thorsten. "Optimizing search engines using clickthrough data." Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002.

[27] G. Xun, X. Jia, V. Gopalakrishnan and A. Zhang, "A Survey on Context Learning," in IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 1, pp. 38-56, Jan. 1 2017.

[28] J. Xuan and M. Monperrus, "Learning to Combine Multiple Ranking Metrics for Fault Localization," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, 2014, pp. 191-200.

[29] Zhong, Hao, et al. "MAPO: Mining and recommending API usage patterns." ECOOP 2009–Object-Oriented Programming (2009): 318-343.

[30] M. Lormans and A. van Deursen, "Can LSI help reconstructing requirements traceability in design and test?," Conference on Software Maintenance and Reengineering (CSMR'06), Bari, 2006, pp. 10 pp.-56.

[31] Hang Li, "Learning to Rank for Information Retrieval and Natural Language Processing," in Learning to Rank for Information Retrieval and Natural Language Processing , 1, Morgan & Claypool, 2011

[32] Paik, Jiaul H. "A novel TF-IDF weighting scheme for effective ranking." Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval. ACM, 2013.

[33] Rehurek, Radim, and Petr Sojka. "Software framework for topic modelling with large corpora." In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. 2010.

[34] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

[35] Tien-Duy B. Le, Richard Jayadi Oentaryo, David Lo: Information retrieval and spectrum based bug localization: better together. ESEC/SIGSOFT FSE 2015: 579-590

[36] Shaowei Wang, David Lo: AmaLgam+: Composing Rich Information Sources for Accurate Bug Localization. Journal of Software: Evolution and Process 28(10): 921-942 (2016)

[37] Inferring Links between Concerns and Methods with Multi-abstraction Vector Space Model. ICSME 2016: 110-121

[38] Tien-Duy B. Le, David Lo, Claire Le Goues, Lars Grunske: A learning-to-rank based fault localization approach using likely invariants. ISSTA 2016: 177-188

[39] Yuan Tian, Dinusha Wijedasa, David Lo, Claire Le Goues: Learning to rank for bug report assignee recommendation. ICPC 2016: 1-10

[40] Bamler, Robert, and Stephan Mandt. "Dynamic word embeddings." International Conference on Machine Learning. 2017.

[41] Jiang, He, et al. "An unsupervised approach for discovering relevant tutorial fragments for APIs." Proceedings of the 39th International Conference on Software Engineering. IEEE Press, 2017.