An Improved Approach based on Balanced Keyword Weight to Traceability Recovery

Xinye Wang, Qinghua Cao and Qing Sun

School of Computer Science and Engineering, Beihang University, Beijing, 100000, China

*E-mail: wangxydhc@163.com

Abstract. Software traceability recovery has become more and more important to the software life cycle recent days with the increasing complexity of software system. However, it is a tedious and error-prone task by manual. Many researchers have devoted themselves in building and predicting traceability links. In this paper, we approach a novel method to calculate text similarity and improve our results by a learning to rank method. And a couple of experiments are conducted to verify our method.

1. Introduction

Requirements traceability is a part of requirements management within software development and systems engineering, defined as the ability to "describe and follow the life of a requirement in both a forwards and backwards direction, and through periods of ongoing refinement and iteration in any of these phases". Traceability ensures that no requirements are overlooked. Especially when certifying safety-critical products it is required that all requirements are realized. Traceability recovery plays an important role in software engineering management, safety analysis and change impact analysis. It is required to be verification and validation by many institutions, such as the USA Federal Aviation Administration (FAA), the USA Food and Drug Administration (FDA). However, the traditional approach to maintaining traceability links is typically achieved either by requirements management tool, or by a spreadsheet or word document directly, which is tedious, time-wasting and error-prone.

Many researchers maintain traceability links by information retrieval (IR) methods [2][4] and machine learning (ML) [3] methods. IR methods like Vector Space Model (VSM), Latent Semantic Analysis (LSA) are usually based on word frequency, regardless of semantic information. Meanwhile ML methods like W2V etc., are turned out to be not suitable in complex text like software artifacts.

In this paper, we present a traceability recovery model that can help us build the traceability links. Firstly, we propose a novel approach to get traceability links by word embedding and tf-idf methods. And then a predict model is generated by learning to rank (LtR) method, which improves the performance in precision of last step. By this way we propose an effective method to get the similarity between two software artifacts, and the introduce of LtR can improve the precision of retrieval results and make our method extensible.

The remainder of this paper is organized as follows: in part II, we introduce some related work relate to our work including IR method in traceability recovery, and LSI, word embedding, and LtR methods. Then in part III and IV, we introduce our method, experiment and analyse the result in detail. Finally, we summarize our work and look into the future work.
2. Related Work

2.1. IR in Software Engineering (SE)
IR techniques have been applied in more and more SE tasks, such as traceability recovery [5, 6], software reuse [1] and feature location [1]. Many researchers devoted themselves this field to improve the retrieval performance.

Dag et al. applied text similarity to requirement traceability links by text similarity [7]. At the same year G. Antoniol et al. built traceability links by Probability Model and (VSM) method [8], both of the methods work well at their task, which inspired our work.

Word embedding, proposed by Mikolov et al., recently has attracted many researchers on text similarity tasks. There are two architectures for training word embedding mentioned: the skip-gram and the continuous bag-of-words (CBOW) [9]. Word was represented as a vector in low dimensional.

Recently, researchers propose novel methods to solve the traceability tasks by word embedding. In [10], X. Ye, etc. calculate document similarities by learning vectors on software documents and cosine of the two words. In their 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.

2.2. Learning to rank
Learning to rank, a supervised or semi-supervised ML method, has been widely used in IR, NLP and many other data mining fields. It is used for bug localization [11, 12], fault location [13] and requirement traceability [4], by making use of the features of artifacts in SE. Given a query \( q \) and a set of corresponding candidate documents \( D_i \), we can get the formula (1).

\[
R = f(w, \phi(q_i, D_i)) \tag{1}
\]

In formula (1), \( q \) is query features extracted from the query, \( D_i \) donates the candidate documents corresponding to \( q \), \( \phi \) is the mapping function, \( w \) donates the weight matrix for the feature, and \( f \) is a ranking approach based on pairwise approach Ranking SVM, which has been proved to be efficient in document tasks.

3. Method

3.1. Method Overview.
There are three main steps in our method: data preparation, similarity calculation and applied the LtR method. In the first step, we process the basic software artifacts to tokens and train the word embedding with the Wikipedia corpus by gensim, a powerful python tool in NLP. Then we calculate similarity by an improved text similarity method. And at last we apply a learning to rank method to predict the remaining links from the generated list.

3.2. Similarity Algorithm

3.2.1. Get IPTs by TF-IDF. Different words account for different proportions of the document information has been proved by [2], and we use the term frequency-inverse document frequency (tf-idf) to describe the weight of the words in the artifacts. It can be calculated by formula (2), (3), (4).

\[
tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2}
\]

\[
idf_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|} \tag{3}
\]

\[
tfidf_{i,j} = tf_{i,j} \times idf_i \tag{4}
\]
In the formula, $tf_{i,j}$ refers to the value of word $t_i$ in document $d_j$, $n_{i,j}$ means the $i_{th}$ word appears $n$ times in the text. $\sum_k n_{k,j}$ is the times of $t_i$ appears in $D$. And $D$ is the documents in dataset, $|\{j : t_i \in d_j\}|$ is the number of the word contained in the documents. The larger the value of idf, the less likely the word occur in other documents. We get tf-idf by multiplying tf times idf. And then we take the top n% words as the key word set called IPTs.

3.2.2. Word Similarity. In this work, similarity between two words will be calculated by cosine similarity (6). However, words are not the same important to the text. To increase the weight of similar words and reduce the weight of less similarity words, we set two thresholds $\delta_l$ ($0 < \delta_l < 0.5$) and $\delta_h$($0.5 < \delta_l < 1$), if any of the words $w_i$ and $w_j$ belongs to IPTs, the similarity of two words can be calculated by (6), where $sim_{mod}$ means the weighted similarity between two words and $sim_{pre}$ stands for the similarity calculated by (5). And $r$ is a harmonic parameter that larger than 1.

$$sim_{w2w}(w_i, w_j) = \cos (w_i, w_j) = \frac{w_i \cdot w_j}{||w_i|| \cdot ||w_j||}$$

$$sim_{mod} = sim_{pre} - \frac{(0.5 - sim_{pre})}{r} \quad (sim_{pre} < \delta_l, \delta_h < sim_{pre}) ~ (5)$$

$$s.t. \quad w_i \in IPTs \ or \ w_j \notin IPTs$$

3.2.3. Text Similarity. In [10], the similarity of word and text can by calculated by (7), which means the max product of the word $w_i$ and each word $w_j$ in text $T$. The similarity of two text can be calculated by (8):

$$sim_{w2t, ori}(w, T) = \max_{w' \in T} sim_{w2w}(w, w')$$

Then we can get the similarity of sentence $T_i$ and $T_j$ by (8):

$$sim_{w2t, asy}(T_i \rightarrow T_j) = \frac{\sum_{w \in P(T_i \rightarrow T_j)} sim_{w2t, ori}(w, T_j)}{|Q(T_i \rightarrow T_j)|}$$

$$s.t. \quad Q(T_i \rightarrow T_j) = \{w \in T_i | sim_{w2t, ori}(w, T_j) \neq 0\}$$

$S$ and $T$ stand for different texts, and $Q(T_i \rightarrow T_j)$ means the similarity between $T_i$ and $T_j$ greater than zero.

Considering that text similarity is independent of text order, and texts are usually not the same length, we take the short sentence as $T_s$, and $T_l$, and get each word in $T_l$ with $T_s$.

3.2.4. Learning to rank. To get more accurate traceability links, besides semantic features, we take the learn to rank method to improve the final result. IR SVM has been proved to be an effective method in document retrieval occasions. The problem can be expressed as shown in formula (9), where $r(\hat{r})$ means the $i_{th}$ rank level of the document pair, $\tau_r(\hat{r})$ stands for the value of $r(\hat{r})$. $d(\hat{d})$ is the document pair relevant to document $i$, $\mu_{d(\hat{d})}$ is the value of $d(\hat{d})$.

$$\min_w \sum_{i=1}^{m} \tau_{r(\hat{r})} \mu_{d(\hat{d})} \left [ 1 - y_i \langle w, x_i^{(1)} - x_i^{(2)} \rangle + \lambda ||w||^2 \right ]^+$$

Feature selection is also an important part of learn to rank, in this paper, we take five features which is frequently used in text retrieval, semantic similarity, Tanimoto, Sum of inverse document frequency(idf), number of key words (tf-idf etc.), and the number of words in the text.

4. Experiment
In this part, we verify the algorithm mentioned in part III. We introduce the data preparation, experiment settings, and the interpretation of the result.
4.1. Dataset Description and Preprocess.
CoEST is a website run by CoEST, an open and prestigious community. It provides some datasets for participation from both academics and industry. We take MODIS, CM1, eTOUR, EasyClinic and iTrust as our dataset, which include not only requirements, but also source codes, test cases, use cases, UML and so on.

There are two kinds of datasets in our dataset: text files, high-level and low-level requirements, use cases, interaction diagrams, test cases, class description, etc. included and code files, both of the traceability links are provided in the datasets.

In this work, we split texts, both of the text files and source code files, into bag of words by whitespaces, remove punctuations, numbers and stop-words. For text files, we take the stop words list provided by nltk, then specific to source code files, we take java and C language keywords as stop-words, which appears frequently in the text. At last we split compound words by upper camel case rule and lowercase the text.

4.2. Experiment Description.
We take LSI and W2V to set up comparative experiments, which are widely used in text similarity. LSI, a method of restoring software tracking links using latent semantic indexing method works effectively in IR [2], W2V [9] works well in requirement traceability in software engineering. Both of the methods are proved to be effective in software engineering. And for our method, called AW-SCM-A, we take the top 30% percent words, and calculate text similarity by the method mentioned in part III. And for the AW-SCM-A model, we perform experiment on the same datasets and compare our approach with ENRL which combinations of 12 different NLP techniques with 4 machine learning classifiers in the traceability task proposed in [3].

4.3. Results.

| DataSet | Link Type | LSI  | W2V  | AW-SCM-A |
|---------|-----------|------|------|----------|
|         |           | PRC  | RE   | PRC  | RE  | PRC  | RE  |
| CM1     | HL→LL     | 0.127| 0.41 | 0.262| 0.217| 0.371| 0.329|
| eTOUR   | UC→CC     | 0.077| 0.221| 0.098| 0.332| 0.088| 0.415|
| MODIS   | HL→LL     | 0.286| 0.332| 0.278| 0.418| 0.255| 0.563|
| iTrust  | UC→CC     | 0.099| 0.45 | 0.192| 0.363| 0.198| 0.322|
| EasyClinic | 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 |

4.3.1. Similarity Algorithm. As shown in Table 1, compared with the LSI method, the AW-SCM-A model similarity algorithm in precision (PRC) is about 33.3% higher and recall (RE) is 24.5% higher. As for W2V method, there is a 6.6% improvement in PRC and 17.2% in RE. Both of W2V and our method are generally superior to the LSI method, which proves that the semantic similarity of the text is efficient in software text.

In figure 1, we can see our method improves the average F1-scores, it works better in all kinds of traceability links on different kind of datasets, such as text to text and text to source code, especially in high-level requirements to low-level requirements, which indicts that tf-idf weight method is available in traceability tasks and our method achieve relatively better results for data without code.
4.3.2. AW-SCM-A Model. In another experiment, we compare our method with the state-of-art method ENRL, conduct the experiment on the same dataset and compare evaluation indicators. The result is shown in table III. We can see the AW-SCM-A method performs better in Mean Relative Error (MRE) on both of the datasets, which indicates that our method generated relatively fewer errors in the links. It proves that our method is practical in traceability tasks.

| Method       | DataSet | MRE   |
|--------------|---------|-------|
| ENRL         | eTOUR   | 0.0290|
|              | EasyClinic | 0.0211|
| AW-SCM-A     | eTOUR   | 0.0208|
|              | EasyClinic | 0.0059|

5. Conclusion and Future Work

In this paper, we proposed the AW-SCM-A model, which combines a text similarity method and the Learn to rank method. We adjust the weight of single word by making use of tf-idf weight. And a couple of experiments have been conducted to prove that our method is practical in all kinds of traceability tasks.

In the future, we would take the word order into consideration, and improve our similarity method. And for the LtR method, we can try more combinations of features. All of that can help improve our result. Besides, we’d like to analyze change impact by our requirement links.

References

[1] Cleland-Huang, Jane, et al. (2014) Software traceability: trends and future directions. In: Future of Software Engineering, Hyderabad, India, 55-69.

[2] Hayes, Jane Huffman, A. Dekhtyar, J. Osborne. (2003) Improving Requirements Tracing via Information Retrieval. In: IEEE International Conference on Requirements Engineering IEEE Computer Society, 138.

[3] D. Falessi, M. D. Penta, G. Canfora, et al. (2017) Estimating the number of remaining links in traceability recovery. In: Empirical Software Engineering,1-32.
[4] Marcus, Andrian, and J. I. Maletic. (2003) Recovering documentation-to-source-code traceability links using latent semantic indexing. In: International Conference on Software Engineering, 125-135.

[5] A. D. Lucia, M. D. Penta, R. Oliveto, et al. (2011) Improving IR-based Traceability Recovery Using Smoothing Filters, In: IEEE 19th International Conference on Program Comprehension, Kingston, 21-30.

[6] Capobianco, Giovanni, et al. (2013) Improving IR-based traceability recovery via noun-based indexing of software artifacts. In: Journal of Software: Evolution and Process, 743-762.

[7] J. N. O. Dag, B. Regnell, P. Carlshamre, et al. (2002) A Feasibility Study of Automated Natural Language Requirements Analysis in Market-Driven Development. In: Requirements Engineering, 20-33.

[8] G. Antoniol, G. Canfora, G. Casazza, et al. (2000) Recovering traceability links between code and documentation. In: International Conference on Software Maintenance, 40.

[9] T. Mikolov, K. Chen, G. Corrado, et al. (2013) Efficient Estimation of Word Representations in Vector Space. In: Computer Science.

[10] Ye X, Shen H, Ma X, et al. (2016) From Word Embedding to Document Similarities for Improved Information Retrieval in Software Engineering, 404-415.

[11] L. B. T. Duy, L. O. David, C. L. Goues, et al. (2016) A Learning-to-Rank Based Fault Localization Approach using Likely Invariants.

[12] Y. Tian, D. Wijedasa, Lo D, et al. (2015) Learning to rank for bug report assignee recommendation. In: IEEE International Conference on Program Comprehension.

[13] J. Xuan, M. Monperrus, (2014) Learning to Combine Multiple Ranking Metrics for Fault Localization, In: 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, 191-200.