Enhancing Automated Requirements Traceability by Resolving Polysemy

Wentao Wang*, Nan Niu*, Hui Liu†, and Zhendong Niu†
* Department of Electrical Engineering and Computer Science, University of Cincinnati, USA
† School of Computer Science and Technology, Beijing Institute of Technology, China
Email: wang2wt@mail.uc.edu, nan.niu@uc.edu, {liuhui08, zniu}@bit.edu.cn

***Abstract*—Requirements traceability provides critical support throughout all phases of software engineering. Automated tracing based on information retrieval (IR) reduces the effort required to perform a manual trace. Unfortunately, IR-based trace recovery suffers from low precision due to polysemy, which refers to the coexistence of multiple meanings for a term appearing in different requirements. Latent semantic indexing (LSI) has been introduced as a method to tackle polysemy, as well as synonymy. However, little is known about the scope and significance of polysemous terms in requirements tracing. While quantifying the effect, we present a novel method based on artificial neural networks (ANN) to enhance the capability of automatically resolving polysemous terms. The core idea is to build an ANN model which leverages a term's highest-scoring coreferences in different requirements to learn whether this term has the same meaning in those requirements. Experimental results based on 2 benchmark datasets and 6 long-lived open-source software projects show that our approach outperforms LSI on identifying polysemous terms and hence increasing the precision of automated tracing.**

***Index Terms*—Requirements traceability, automated requirements tracing, polysemy analysis, term coreference**

I. Introduction

Requirements traceability is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction" [1]. It has been shown that requirements traceability is vital for other activities throughout all phases of software engineering, such as program comprehension [2], change impact analysis [3], [4], requirements reuse [5], [6], software maintenance [7], refactoring [8], test generation [9], and verification and validation (V&V) [10].

In practice, requirements traceability can be achieved by managing trace links between software artifacts like from high-level to low-level requirements. Previous research found that developers do not build the trace links to the proper level of detail or at all [11]. Therefore, activities like V&V are faced with the time consuming, mind-numbing, effort-intensive, and error-prone task of manually creating trace links in the "after the fact" way, i.e., traceability information is not created as a process unfolds [11].

To overcome limitations of the manual approach, information retrieval (IR) methods have been introduced for automated trace links generation [11], [12], [13], [14], [15]. IR-based approaches take a requirement as the trace query and return artifacts with high textual similarities as candidate links. Previous research has demonstrated that acceptable recall levels (percentage of correct links in answer set that have been retrieved) over 90% can often be achieved only at extremely low levels of precision (percentage of retrieved correct links over all the retrieved links) [11], [16], [17].

After manually analyzing 6 open-source software (OSS) projects, we found that 52.67% of the incorrect links that are retrieved contain polysemy (i.e., one word has multiple meanings). To illustrate this, we draw on an example from the OSS project JBoss Transaction Manager (Narayana or JBTM)[1]. JBTM is a transactions toolkit which provides support for applications developed using a broad range of standards-based transaction protocols (e.g, web-service transactions and REST transactions). Terms "subsystem", "host", and "configuration" in the requirement JBTM-1644 [18] stating that: "Moving the file based object store to another host and started another application server with the same version and configuration. Identical configurations probably isn't necessary as long as the database subsystem is configured correct and the transaction component is configured in the same way", indicate "database system", "file storage", and "configuration for database" respectively. However, in the enhancement JBTM-1706 [19] stating that: "Make sure that RTS subsystem is configured correct and chooses the correct Undertow host instead of requiring default Undertow configuration", they represent "JBTM component RTS", "web server Undertow", and "configuration for Undertow" respectively. These two artifacts would likely be mistakenly linked together by IR-based automated tracing approaches, such as the vector space model (VSM), which represent all artifacts as bags of words and therefore fail to recognize the artifacts' polysemy information.

Among popular IR-based automated tracing approaches, only latent semantic indexing (LSI) is considered to be able to resolve the polysemy problem [20], [21]. For a given document collection, LSI creates a term-by-requirement matrix, and then applies a mathematical technique called singular value decomposition (SVD) to the matrix to construct a reduced space, called LSI space. By using LSI space, it hopes to preserve important latent semantic structures of documents, as well as to remove "noise". One target noise could be the less important meanings of polysemous terms. Deerwester *et al.* [22] pointed out that LSI offers a partial solution to the polysemy problem, and its failure to alleviate polysemy in LSI comes from the fact that a polysemous term with

[1] http://narayana.io/index.html

2332-6441/18/$31.00 ©2018 IEEE
DOI 10.1109/RE.2018.00-53

RE 2018, Banff, Canada
Research Paper

multiple meanings is represented as a single point in the space. Deerwester *et al.* [22] also pointed out that a better solution for polysemy problem is to have an automated way to detect the fact that a particular term has several distinct meanings and to place this term in several points in the space.

On the other hand, artificial neural networks (ANN) techniques have been successfully applied to solve many natural language processing (NLP) problems. In this paper, we propose an ANN-based approach to enhance the capability of IR-based requirements tracing by identifying polysemous terms in requirements. The core idea is to train an ANN model which leverages a term's highest-scoring coreferences in different requirements to learn whether this term has the same meaning in those requirements. We compared our newly proposed approach with LSI to evaluate their abilities to resolve polysemy. The results showed that our approach outperforms LSI, thereby enhancing the accuracy of requirements trace link recovery. Finally, we detected the importance of input features included in ANN model via a feature ablation study. Surprisingly, stakeholder features (e.g., whether requirements are reported by different stakeholders) did not significantly improve the performance of the ANN model. We argue that stakeholders with similar background share the terminology, and therefore tend to use the same terms consistently when describing the objects.

This paper makes 3 main contributions, including investigating the impact of polysemous terms in requirements tracing, proposing an approach based on ANN model to enhance IR-based requirements tracing, and testing the performance of our approach on 2 benchmark datasets and 6 OSS projects. The rest of the paper is organized as follows. Section II provides background. Section III presents quantifying analysis of polysemy's effect in requirements tracing. Section IV introduces our polysemy enhaced approach. Section V describes experimental design and results. Section VI discusses related work, limitations, and implications. We conclude the paper and suggest potential future research directions in Section VII.

II. Background

The representation of a set of documents as vectors in a common vector space is known as the VSM and is fundamental to a host of IR operations ranging from scoring documents on a query, document classification, and document clustering [23]. In this section, we first review core ideas of VSM and another IR method, LSI, which is considered to be able to resolve polysemy.

In VSM, let $T = \{t_1, t_2, \cdots, t_N\}$ denote all the terms in the given document collection. A document d is represented as a vector $[w_1, w_2, \cdots, w_N]$ of term weights. The frequency of term t_i in the document is often used to assign weight to w_i. However, the term frequency might represent the bias toward long-text documents or frequent words in the corpus. To mitigate this risk, term frequency and inverse document frequency (TFIDF) is used to calculate term weights:

$$w_i = tf_{t_i,d} \cdot idf_i \tag{1}$$

where $tf_{t_i,d}$ is the frequency of t_i in d, and idf_i is computed as $idf_i = \log \frac{M}{df_i}$. M is the total number of documents and df_i is the number of documents containing t_i.

For a given query q and a document d, the standard way of quantifying their similarity is the normalized inner product between their vector representations, called cosine similarity. The basic tenet is that the same term is used to express the same meaning, and documents share more terms are more similar than documents with only a few terms in common. Since some terms have multiple meanings, polysemous terms in a user's query will literally match terms in documents that are not relevant to the user's information need, thus causing rapid degradation of precision [22].

Instead of retrieving information by literally matching terms in the documents, LSI tries to overcome the polysemy problem by using statistically derived conceptual indices. LSI assumes that there is some underlying or latent structure in term usage that is partially obscured by variability in term choice [22]. A particular statistical technique SVD is used to estimate this latent structure, and get rid of the obscuring "noise." Retrieving is then performed on documents and query's "latent semantic structure" representation.

For a given document collection, LSI first constructs a term-by-requirement matrix (A). Each row represents a document, and a term is represented as a column. The value of a cell A_{ij} is the number of times term t_i appears in the document d_j. SVD is applied to decompose A into a factorization of the form: $A = USV^T$. Here, S is a diagonal matrix of eigenvalues of A^TA, and U and V are called term matrix and document matrix respectively. Dimensionality reduction is achieved by replacing S with a matrix S_k, which consists of the top k diagonal elements of S. The reduced matrix $A_k = U_k S_k V_k^T$ is used in place of A. SVD analysis allows reduced space to keep major associative patterns of term usages, and ignore less frequent term usages which can be described as less important meanings of certain terms. However, it works while only one usage pattern or meaning of a polysemous term frequently appears in the give document collection. If the frequencies of two or more meanings of a particular polysemous term have no significant difference, the reduced space actually represents the weighted average of different meanings of this term. Therefore, using the reduced space will create a serious distortion, and cause more imprecision on the retrieval results [22].

LSI has been employed in a wide range of software engineering activities such as categorizing source code files [24] and detecting high-level conceptual code clones [25]. Marcus and Maletic [26] have applied LSI to recover traceability links between documentation and source code. The results based on two datasets are promising enough to warrant future research. They also believed that other information is needed to improve the performance of LSI [26]. In fact, Deerwester *et al.* [22] pointed out that "what is needed is some way to detect the fact that a particular term has several distinct meanings and to subcategorize it and place it in several points in the space."

Another approach that can be used to detect polysemy is

TABLE I
SIX PROJECTS USED TO ANALYZE THE IMPACT OF POLYSEMY IN AUTOMATED REQUIREMENTS TRACING. TRACE LINK TYPES: ① REQ.-TO-ENHANCEMENT; ② REQ.-TO-SUB REQ. ③ TASK-TO-SUB TASK ④ REQ.-TO-TEST

Project	Characteristics					Statistics				
	domain	written in	initial release	latest release	trace link types	# req.s	% of req.s have links	avg. # links per req.	# terms per req.	polysemy in req.s
AIRFLOW	workflow execution	Python	Oct 16 2014	Feb 02 2017	① ② ③ ④	629	20.99%	0.26	52.73	4.27 (8.10%)
ANY23	RDF data extraction	Java	Jul 16 2012	Feb 26 2017	① ② ③ ④	182	32.97%	0.41	37.61	3.04 (8.08%)
DASHBUILDER	data reports	Java HTML	Aug 27 2014	Apr 14 2016	① ③	85	5.88%	0.11	48.37	4.73 (9.78%)
DROOLS	business rules	Java	Nov 13 2012	Jul 17 2017	① ② ③	486	27.16%	0.31	57.82	6.21 (10.74%)
IMMUTANT	complexity reduction	Clojure	Mar 14 2012	Jun 23 2017	① ② ③	404	16.09%	0.20	29.97	2.53 (8.44%)
JBTM	business process	Java C++	Dec 05 2005	Jul 14 2017	① ② ③	1575	52.57%	0.72	49.74	4.43 (8.90%)

part-of-speech (POS) tagging [27]. POS refers to the syntactic roles of terms in the sentences (e.g., nouns and verbs). In POS tagging augmented VSM (VSM-POS), only terms which belong to a particular part of speech are considered in vector space. Recent research reveals nouns and verbs carry higher information values than other parts, capturing main actions and objects in software artifacts [28], [29], [30], [31], [32]. Therefore, two specific forms of VSM-POS, nouns only (VSM-POS-N) and verbs only (VSM-POS-V), can be applied to improve the accuracy of VSM.

III. POLYSEMY QUANTIFYING ANALYSIS

Before we introduce our polysemy enhanced approach, we first describe a manual analysis on 6 OSS projects to show the impact of polysemy in automated requirements tracing. TABLE I lists the basic characteristics and statistics of the 6 projects. All 6 projects use the issue tracking system Jira[2] to manage their requirements and other textual artifacts. Stakeholders have defined relationships between different artifacts in Jira systems. While traceability refers to dependencies among artifacts that are part of a single work product within the software development process, we focus on only the relationships from high-level artifacts (e.g., requirements and tasks) to low-level artifacts (e.g., test and sub-tasks). Trace link types in 6 OSS projects are listed in TABLE I.

Jira systems record not only user operations (e.g., creating requirements, linking related issues), but also timestamps of all operations. Therefore, operation histories can be retrieved. From the analysis of historical data on Jira systems, we know that stakeholders do not always provide complete links. In fact, an average of 67.19% trace links in the 6 projects was created at least one day (24 hours) after requirements were created. In other words, most trace links were discovered during implementation phrase. About 12% links were created after the requirements were closed. On the other hand, stakeholders rarely modify or delete trace links. Historical data shows that, on average, only 4.32% trace links in 6 projects have been modified or deleted. We argue that stakeholders provide trace links that they feel confident only. Therefore, we can use those trace links as answer sets. We selected these 6 projects mainly for the following reasons.

First, we select the 6 projects from 345 OSS projects. Our guiding principle of choosing the representative projects is to find projects with different sizes and active time durations. In addition, the 6 projects cover all trace link types in 345 OSS projects. This increases our confidence about the representativeness of 6 projects.

Second, the 6 projects are in different domains. A set of terms have specific meanings in those domains. For instance, "table" in the domain "business rules" indicates "decision table" while in another domain "database", it indicates "database table". Selected projects can help to test the performance of our approach on distinguishing terms' domain-specific meanings.

Third, the chosen projects are implemented in different programming languages. A particular term may have specific meanings in programming languages. For instance, in HTML, "tag" indicates "HTML tag". However, the most common explanation of it is "a label attached to someone or something". Selected projects can help to test the performance of our approach on distinguishing terms' programming language-specific meanings with their general meanings.

In addition, with the change of time, the meanings of terms may also evolve. All 6 OSS projects are long-lived projects. They can help to test our approach's capability of catching terms' meaning evolution over the time.

Finally, the more terms of a requirement, the more polysemous terms it tends to contain. The average length of requirements (number of different terms per requirement; the second rightmost column in TABLE I) is different in these 6 projects. The maximum (57.82 in DROOLS) is almost twice of the minimum (29.97 in IMMUTANT). If our approach works well for different requirements lengths, it will increase our confidence on applying our approaches to other software artifacts such as source code.

IR-based trace link recovery approaches rank all links by

[2]https://www.atlassian.com/software/jira

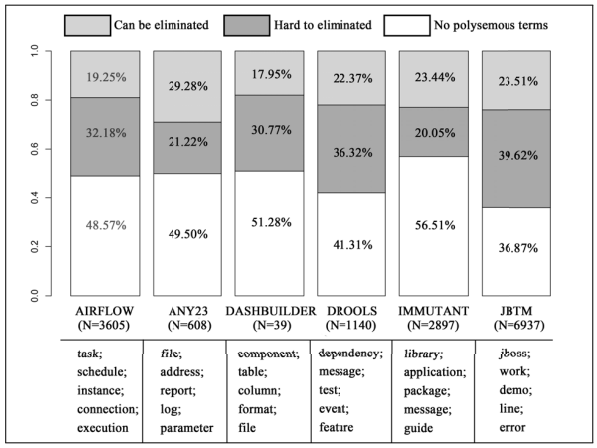

Fig. 1. False positives in automated tracing: N is the total number of false positives in a project, white part shows the percentage of false positives that do not contain polysemy; light gray part shows the percentage of false positives that can be ruled out from returned lists after resolving polysemy; dark gray part shows the percentage of incorrect links that cannot be removed from retrieved results after resolving polysemy.

their similarities with a requirement. Only top N trace links are returned to developers. To study the impact of polysemy on IR-based tracing, we applied VSM on all requirements which have been linked to other artifacts. For each requirement, we cut the rank list at the point where all correct links are retrieved, i.e.,100% recall. All requirements and retrieved links were then analyzed by two authors of this paper independently to find all polysemous terms (i.e., the terms which have different meanings in requirements and false positives). Two authors have at least 3 years working experience on OSS projects. They started research on the 6 OSS projects from May 2017 and conducted many research and development tasks on those projects. Therefore, they are experts with necessary domain knowledge about the 6 projects. We use Fleiss' kappa κ [33] to measure inter-rater agreement between the two experts. The result ($\kappa = 0.57$) shows that there is a "good" agreement between the two experts [34]. For the terms that experts had different opinions, the experts made the final judgement through a face-to-face discussion. Polysemous terms identified manually by experts are used not only in polysemy analysis, but also as training set for our approach. We also acknowledge that, manually detecting polysemous terms is a time consuming task (two experts spent two weeks to label all polysemous terms in 6 projects). The results may still contain errors. Additional information like project glossary can be used to support manual analysis. The rightmost column in TABLE I shows experts' analysis results. The values show average numbers of different polysemous terms in each requirement. Percentages in brackets represent proportions of polysemy in each requirement. We find that the proportions of polysemy are very similar in 6 projects (i.e., from 8.44% to 10.74%). This finding confirms our previous conjecture that longer text contains more polysemous terms.

We then separate those polysemous terms as different terms while building term-by-requirement matrix. For example, if term t is a polysemous term in a requirement, we mark it as t_1 in the requirement, and t_2 in the retrieved links. Solving polysemy could reduce similarity scores between requirements and incorrect links (false positives), thus improve precision of the results. Therefore, our analysis focuses on only false positives. The results of updated term-by-requirement matrix show that an average of 22.63% false positives in the 6 projects are eliminated from the returned lists. Fig. 1 illustrates the results of 6 projects. For example, in AIRFLOW, 3605 false positives were retrieved by using original term-by-requirement matrix, 48.57% of which do not contain polysemy (white part), and 19.25% false positives can be ruled out by using updated term-by-requirement matrix (light gray part). However, there are 32.18% false positives which contain polysemous terms that are still retrieved (dark gray part) after separating polysemous terms. We observed the following two reasons why certain false positives had not been resolved after separating polysemous terms: 1) Some requirements and their correct links do not share any terms, thus their similarity scores are 0. It is hard to reduce the similarity scores of requirements and false positives below 0 by addressing only the polysemy problem; and 2) The weights of some polysemous terms are relatively small. They do not have the significant impact on decreasing similarity scores. Separating polysemous terms combined with a good way to adjust their weights may further reduce the similarity score, thereby helping rule out false positives.

For each project, we also list top 5 most frequently occurred polysemous terms. For example, in AIRFLOW (a workflow management system), the top 5 polysemous terms are "task", "schedule", "instance", "connection", and "execution", which mean "workflow task" or "development task", "a plan" or "arrange a event", "workflow task" or "SQL operation", "dependency between workflow tasks" or "database connection", and "execute workflow task" or "execute SQL query" respectively.

In summary, an average of 52.66% incorrect links in the 6 OSS projects is retrieved containing polysemy. Therefore, we argue that polysemy is a serious problem which leads to lower precision. Successfully resolving the polysemy problem can significantly improve the precision of automated requirements tracing. In the next section, we describe our approach of identifying polysemous terms in requirements.

IV. Polysemy Detection Based on Artificial Neural Networks

In NLP, the phenomenon that two terms refer to the same real-world entity is called coreference [35]. For instance, the linguistic expressions "subsystem" and "database" refer to the same entity in the requirement mentioned in Section I. The resolution of coreference has been referred to as calculating semantic similarity between terms [36], [37]. Coreference can be used to resolve the polysemy problem. The assumption is that, if one polysemous term in different artifacts refers to different entities, this term has different meanings in those artifacts. For instance, term "subsystem" in the enhancement

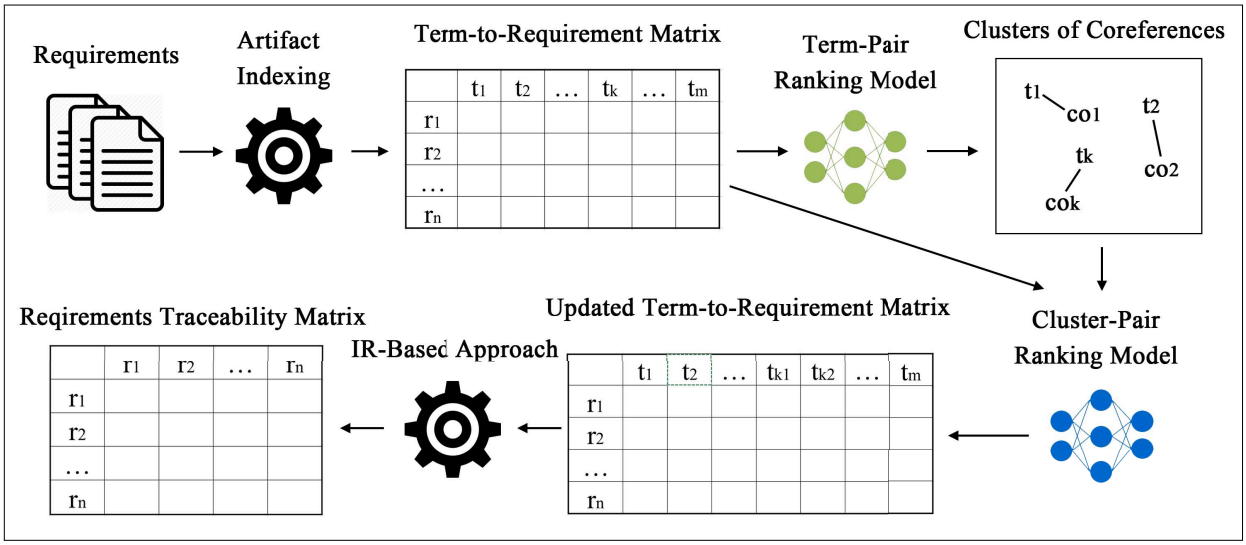

Fig. 2. System architecture of polysemy enhanced automated requirements tracing

refers to "JBTM component RTS"; not "database". In this section, we describe the system architecture and technical details of our polysemy enhanced requirements tracing approach.

A. System Architecture

Our polysemy detection approach follows the process depicted in Fig. 2. The given requirements are first fed into artifact indexing which includes two steps (tokenization and stop words removal). The output of artifact indexing is a term-by-requirement matrix.

Our term-pair ranking model scores pairs of terms in the same requirement by passing their distributed representations through a feedforward neural network (FNN) which contains one input layer, one output layer, and three hidden layers. FNN is a fundamental type of ANN. In FNN, the information moves in only one direction, i.e., from the input layer to the output layer. FNN is capable of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense [38]. In other words, FNN can be used to learn any non-linear relationships. Therefore, it has been viewed as a powerful method that can be applied to many machine learning tasks [39]. Previous research demonstrated that FNN can provide satisfactory results on NLP tasks, such as coreference [37] and question answering [40]. The input of term-pair ranking model is a particular term t and the other terms in the same software artifact. The output of our term-pair ranking model is a term pair that contains t and its highest scoring coreference in the same software artifact.

An innovation of our approach is using the cluster-pair ranking model to detect term coreference. Most approaches in NLP detect coreference by linking pairs of terms which have high semantic similarities together [37], [41], [42], called term-level coreference. The similarity of two terms is usually calculated by using terms and their preceding and following terms. The size of preceding and following terms directly affects accuracies of approaches. For instance, like the state-of-the-art approach [37], by taking two previous and two following words of the term "subsystem" in the requirements and the enhancement mentioned in Section I, we will get "the database subsystem is configured" and "that RTS subsystem is configured". Since 3 out of total 7 terms are shared by these two sub-sentences, term-level coreference may still mistakenly mark "subsystem" in two artifacts as referring to the same entity. On the other hand, increasing the size of preceding and following terms may result in increasing computational complexity. An alternative approach is to treat a term and its highest-scoring term-level coreferences in different artifacts as clusters, and then the semantic similarity of two clusters is used to decide whether this term has different meanings in those artifacts. In the above example, "subsystem" belongs to different clusters (i.e., {subsystem, RTS} and {subsystem, database}), indicating the different meanings that "subsystem" has.

Thus, we design our cluster-pair ranking model in two steps. First, we pass term t and its highest-scoring coreferences in r_i and r_j generated by term-pair ranking model through a pooling layer and a single neural network layer. Then, if the output is false (i.e., t in r_i and r_j refers to different objects, for instance, "RTS subsystem" and "database subsystem"), the term-by-requirement matrix is updated by separating the column of t into two different columns. We believe our cluster-pair ranking model improves the accuracy of coreference detection, thereby, increasing the quality of term-by-requirement matrix.

Finally, IR-based trace link recovery approaches calculate similarities of requirements pairs by using updated term-by-requirement matrix. Only requirements pairs with similarity scores higher than the certain threshold are considered to be related to each other. In the following sub-sections, we will describe the neural network based term-pair ranking model

and cluster-pair ranking model and their training details.

B. Term-Pair Ranking Model

Our term-pair ranking is based on a FNN model. The structure of the FNN makes it a universal function approximator, which has the capabilities of approximating any continuous function [43]. Therefore, it is suitable to solve a wide range of problems, including clustering and classification [44]. Compared to the single layer perceptron (SLP), the multilayer perceptron (MLP), which includes one or more hidden layers in between an input and an output layer, is more suitable for learning non-linear separable patterns and most data in the real world is non-linear [45].

Term t may appear in different part of a requirement r. We call each copy of t in r a mention of t. Given a mention m of t and its candidate antecedent a in r, the term-pair ranking model produces a score $s_m(a, m)$ representing their compatibility for coreference with a MLP. In this section, we will describe the design of MLP, including the structure of input layer, activation functions of hidden layers, the output layer, and training details.

Input layer: An important input feature is word embedding. Instead of representing terms as single weight, NLP techniques model a term's meaning by embedding it into a high dimensional vector space. The vector is computed from not only the term itself, but also the distribution of terms around it. Therefore, for a particular term, its embedding representations under different contexts are different. Vector model of meaning is considered as the most common way to compute semantic difference [46]. We combine embeddings with other features which are also considered to be able to detect polysemy, and then feed them into FNN models to learn whether a term in different artifacts has different semantic meanings. The detailed information of input features is shown as follows:

- *Embedding features* include word embeddings of the mention, two preceding words, and two following words the mention, average word embeddings of the five preceding words, five following words of the mention, and all words in mention's sentence. We can consider the combination of those embeddings as a specific embedding for a term under the certain context. We initialized our word embeddings with 50-dimensional vectors produced by *word2vec* [47] in Java by Eclipse DL4J[3].
- *Additional features* include the POS tag of the mention (e.g., pronoun, nominal, verb)[4], the mention's position (i.e., index of the mention divided by the number of mentions in the requirement).
- *Distance features* include the distance between the mention and its candidate antecedent in sentences, the distance between the mention and its candidate antecedent in intervening mentions. Like [37], The distance features are binned into one of the buckets [0,1,2,3,4,5-7,8-15,16-31,32-63,64+] and then encoded in a one-hot vector (the value of each vector component is either 1 or 0).
- *String matching features* include head match, exact string match, and partial string match.

In Clark and Manning's approach [37], a mention could be a term or a phrase. However, we assume that a term is a single word. Therefore, features of phrases (e.g., word embedding of the head word, average word embeddings of all words in the phrase, etc.) are not suitable for our term-pair ranking model. Since in term-pair ranking model, we focus on only term-pairs in the same requirement, stakeholder features (e.g., whether the mentions come from the same stakeholder or different stakeholders) of a mention and its candidate antecedent are the same, and adding them will make no difference on the output of the term-pair ranking model. In fact, they are more useful for the cluster-pair ranking model which learns whether a given term t in two different requirements refers to the same object.

Hidden layers: The input gets passed through three hidden layers of rectified linear (ReLU) neurons [49]. Each neuron in a hidden layer is fully connected to the previous layer. The vector $\mathbf{h_i}(a, m)$ represents the output of the ith hidden layer, whose size is M_i :

$$\mathbf{h_i}(a, m) = max(0, \mathbf{W_i}\mathbf{h_i}(a, m) + \mathbf{b_i}) \quad (2)$$

where $\mathbf{W_1}$ is a $M_i \times I$ weight matrix, $\mathbf{W_2}$ is a $M_2 \times M_1$, and $\mathbf{W_3}$ is a $M_3 \times M_2$ matrix. Similar to Clark and Manning [37], we set our hidden layers' sizes to $M_1 = 1000, M_2 = M_3 = 500$.

Compared to other activation functions, such as sigmoid [50], ReLU allows efficient and effective training of FNN on large and complex data. Recent research done by Glorot *et al.* [51] also showed that ReLU is remarkably suitable for naturally sparse data (e.g., term coreference).

Output layer: There is only one neuron in the output layer which is fully connected to the 3rd hidden layer. It produces a score $s_o(a, m)$ to a mention m and its candidate antecedent a representing their compatibility for coreference:

$$s_o(a, m) = \mathbf{W_o}\mathbf{h_3}(a, m) + \mathbf{b_o} \quad (3)$$

where $\mathbf{W_o}$ is a $1 \times M_3$ weight matrix. The term-pair ranking model links each term with its highest scoring candidate antecedent.

Training details: We train our model using backpropagation algorithm [45] with the slack-rescaled max-margin training objective [37], [52]. Suppose the training set consistst of N mentions $\{m_1, m_2, ..., m_N\}$. Let $\mathcal{A}(m_i)$ denote the set of candidate antecedents of a mention m_i, and $\mathcal{T}(m_i)$ denote the set of antecedents which are semantically close to m_i. $\hat{t}_i$ is the highest scoring antecedent of mention m_i in $\mathcal{T}(m_i)$. Then, the goal of training is to minimize the loss which is given by:

$$\sum_{i=1}^{N} \max_{a \in \mathcal{A}(m_i)} \Delta(a, m_i)(1 + s_o(a, m_i) - s_o(\hat{t}_i, m_i)) \quad (4)$$

[3]DL4J or Deeplearning4j is an open-source deep-learning library written for Java and Sala https://deeplearning4j.org.

[4]The Stanford CoreNLP [48] is used to detect POS tags.

where $\Delta(a, m_i)$ is defined as:

$$\Delta(a, m_i) = \begin{cases} \alpha_{FN}, & a = NA \wedge \mathcal{T}(m_i) \neq NA \\ \alpha_{FA}, & a \neq NA \wedge \mathcal{T}(m_i) = NA \\ \alpha_{WL}, & a \neq NA \wedge a \notin \mathcal{T}(m_i) \\ 0, & a \in \mathcal{T}(m_i) \end{cases} \quad (5)$$

According to [37], $(\alpha_{FN}, \alpha_{FA}, \alpha_{WL}) = (0.8, 0.4, 1.0)$ can achieve optimal results. Our stop criteria for training is when we find the same value of equation (4) in different epochs (different rounds of backpropagation) or the total number of epochs is greater than 200.

In conclusion, the purpose of the term-pair ranking model is that, for a particular term in a requirement, finding the most related terms inside the same requirement. For instance, in Section I's example, we embed features of "subsystem" and one of other terms in the requirement (e.g, database) into a vector (input layer). Then this vector is fed into a pre-trained FNN model (i.e., three hidden layers and output layer). The result of the FNN model is a score representing semantic similarity of two input terms. Ideally, among all the terms, the term-pair ranking model will give cluster {subsystem, database} the highest score in the requirement, and will also assign the highest score to cluster {size, RTS} in the enhancement. Then, those clusters are fed into the cluster-pair ranking model to detect whether the term "subsystem" has different meanings in these two software artifacts.

C. Cluster-Pair Ranking Model

Given a term t and two different requirements r_i and r_j, we can obtain two clusters $c_i = \{a_i, m_i\}$ and $c_j = \{a_j, m_j\}$ where a_i and a_j are t's highest-scoring retrieved antecedents in r_i and r_j respectively, and m_i and m_j are two mentions of t in r_i and r_j which were linked to a_i and a_j in term-pair ranking model. Our cluster-pair ranking model produces a boolean score representing whether t in two clusters refers to the same object.

Input: The input of cluster-pair ranking model is the intermediate output of term-pair ranking model called term-pair representation, i.e., the output of the 3rd hidden layer $\mathbf{h_3}$. Given two clusters $c_i = \{a_i, m_i\}$ and $c_j = \{a_j, m_j\}$, the cluster-pair ranking model first combines the information of four term-pair representations $\mathbf{R_c}(c_i, c_j) = [\mathbf{h_3}(a_i, m_i), \mathbf{h_3}(a_i, m_j), \mathbf{h_3}(a_j, m_i), \mathbf{h_3}(a_j, m_j)]$.

Pooling: Pooling is then used to achieve more compact representations, and better robustness to noise and clutter by preserving task-related information while removing irrelevant details [53]. Similar to [37], our pooling operation concatenates the results of max-pooling and average-pooling:

$$\mathbf{r_c}(c_i, c_j)_k = \begin{cases} \max\{\mathbf{R_c}(c_i, cj)_k\}, & for\ 0 \leq k < d \\ avg\{\mathbf{R_c}(c_i, c_j)_k\}, & for\ d \leq k < 2d \end{cases} \quad (6)$$

where $d = M_3 = 500$. Pooling produces a vector representation of cluster-pair (c_i, c_j).

Cluster-pair compatibility for coreference: In term-pair ranking model, all terms are from the same stakeholder (i.e., the requirement creator), and therefore, no stakeholder features are considered. In cluster-pair ranking model, c_i and c_j may come from different requirements created by different stakeholders. Therefore, stakeholder features shall be considered while computing cluster-pair compatibility for coreference. Two stakeholder features are added to vector representations produced by pooling: 1) whether r_i and r_j were created by the same stakeholder; and 2) whether one cluster contains the name of the creator of another cluster. This can be determined by exploring creators' information saved in issue tracking system and/or string matching rules from Raghunathan *et al.* [54].

The updated cluster-pair representation is then fed into a single fully connected layer of size one to produce a score presenting cluster-pair's compatibility for coreference:

$$s_c(c_i, c_j) = \mathbf{W_c}\mathbf{r_c}(c_i, c_j) + b_c \quad (7)$$

Decision making: We defined a policy network π that assigns a probability of c_i and c_j are coreferences:

$$\pi(c_i, c_j) \propto \exp(s_c(c_i, c_j)) \quad (8)$$

Suppose r_j was created later than r_i, the policy network also assigns a score representing mention m_j has no antecedent:

$$\pi(NA, m_j) \propto \exp(s_{NA}(m_j) \quad (9)$$

where

$$s_{NA}(m_j) = \mathbf{W_{NA}}\mathbf{h_3}(NA, m_j) + b_{NA} \quad (10)$$

if and only if $\pi(c_i, c_j) \geq \pi(NA, m_j)$; otherwise, we mark t term in r_i and r_j as the same term, otherwise, t indicates two objects t_1 and t_2. It is noteworthy that only $\pi(NA, m_j)$ is used in decision making, since it is reasonable and relatively easy to detect whether a new requirement refers to existing requirements, other than to predict whether future unknown requirements refer to the current requirement. In addition, we apply the same training strategy described in Section IV-B to train π.

We then use the output of our cluster-pair ranking model to update term-by-requirement matrix. For instance, in Section I's example, if the result cluster-pair ranking model indicates that "subsystem" in the requirement (i.e., JBTM-1644) and the enhancement (i.e., JBTM-1706) refers to different entities, we could separate "subsystem" into two columns in term-by-requirements matrix, "$subsystem_1$" and "$subsystem_2$", and set weights of "$subsystem_1$" in the requirement and the enhancement as $tf_{subsystem,1644} \cdot idf_{subsystem}$ and 0 respectively. Similarly, we also set weights of "$subsystem_2$" in the requirement and the enhancement as 0 and $tf_{subsystem,1706} \cdot idf_{subsystem}$ respectively. The updated term-by-requirement matrix is incorporated into IR-based approaches in order to improve the precision of the trace retrieval results. In the next section, we evaluate our approach with 6 OSS projects and 2 benchmark datasets. The results show that our approach can significantly improve the precision.

TABLE II
COMPARISON OF STANDARD IR-BASED APPROACH AND ITS IMPROVEMENTS (VSM-POLYSEMY VS. VSM-POS-N VS. VSM, LSI-POLYSEMY VS LSI-POS-N VS. LSI): $*$ AND $\circ$ INDICATE THAT THE PERFORMANCE IMPROVEMENT IS STATISTICALLY SIGNIFICANT ACCORDING TO THE NON-PARAMETRIC WILCOXON SIGNED-RANK TEST AT 0.05 AND 0.01 LEVELS RESPECTIVELY; R: RECALL; P: PRECISION

Projects	VSM			VSM-POS-N			VSM-Polysemy			LSI			LSI-POS-N			LSI-Polysemy		
	R	*P*	F_2	*R*	*P*	F_2	*R*	*P*	F_2	*R*	*P*	F_2	*R*	*P*	F_2	*R*	*P*	F_2
AIRFLOW	.78	.11	.35	.76	.15	.42	.88 $*$	.39 $*$	.70 $*$	.79	.21	.51	.78	.26	.56	.91$*$	.39 $*$	.72 $*$
ANY23	.87	.12	.39	.87	.17	.48	.94	.38 $*$	.73 $*$	.89	.20	.53	.87	.21	.53	.96 $\circ$	.37 $*$	.73 $*$
DASHBUILDER	.77	.21	.50	.76	.24	.53	.80	.22	.52	.79	.21	.51	.77	.22	.51	.80	.21	.51
DROOLS	.62	.25	.48	.57	.31	.49	.69	.42 $*$	.61 $*$	.64	.26	.50	.62	.32	.52	.74 $\circ$	.43 $*$	.65 $\circ$
IMMUTANT	.75	.26	.54	.74	.28	.56	.75	.36 $*$	.62 $*$	.76	.28	.57	.74	.33	.59	.78 $*$	.42 $\circ$	.67 $\circ$
JBTM	.73	.17	.44	.69	.21	.47	.79	.33 $*$	.62 $*$	.75	.20	.50	.72	.24	.51	.82 $\circ$	.37 $\circ$	.66 $\circ$
MODIS	.83	.21	.52	.83	.22	.53	.87	.32 $*$	.65 $*$	.85	.23	.55	.85	.24	.56	.91 $*$	.33 $*$	.67 $*$
CM-1	.91	.22	.56	.90	.23	.57	.91	.30 $*$	.65 $*$	.90	.26	.60	.90	.28	.62	.93	.34 $*$	.69 $*$

V. EXPERIMENTAL DESIGN AND RESULTS

A. Experimental Design

Not only were the 6 OSS projects listed in TABLE I used to conduct the experiments, we also used 2 benchmark projects. The MODIS dataset contains 19 high-level and 49 low-level requirements for NASA's Moderate Resolution Imaging Spectrometer. The CM-1 dataset has 235 high-level requirements and 220 low-level design documents for a NASA scientific instrument. The correct trace links are defined by projects' original developers. Both MODIS and CM-1 are available on Promise Website [55].

We first train and test our approach with 6 OSS projects. Again, polysemous terms identified by experts while analyzing polysemy in Section III are used to train our FNN models. For each project, we apply 10-fold cross-validation to evaluate the performance of our models. The primary goal of our models is to automatically update term-by-requirement matrix with the hope that all vector space based models, such as VSM and LSI, can benefit from polysemy analysis. Therefore, we compare our polysemy enhanced automations (VSM-Polysemy and LSI-Polysemy) with the standard approaches (VSM [12] and LSI [26]). Three most frequent and basic measures in IR field (recall and precision) are used to analyze the experimental results. Another important measure is F-measure which is the harmonic mean of recall and precision. Automated tracing methods emphasize recall over precision [11], therefore, F_2 is also used to compare the performances of approaches.

In order to evaluate and improve our feature selection, we perform the feature ablation study on all five feature groups (i.e., word embedding, additional, distance, string matching, and stakeholder features). The feature ablation study is designed to assess the informativeness of a feature group by quantifying the change in predictive power when comparing the performance of an approach trained with all the feature groups versus the performance without a particular feature group. The results of feature ablation can be used to find optimal feature groups by removing feature groups which have negative impact on the approach's performance.

Since manually identifying polysemous terms is a time consuming task and the results may contain errors which can cause inaccuracy of FNN models, an optimal solution for FNN models is to train models with a fully discussed training set with the hope that the trained models can successfully apply to other projects. Therefore, we perform a cross-project testing, e.g., training our models with 6 OSS projects and testing our approach with 2 benchmark projects.

B. Results

We compare polysemy enhanced automated requirements tracing approaches with the baseline approaches, i.e., VSM-Polysemy vs VSM-POS-N vs. VSM, LSI-Polysemy vs LSI-POS-N vs. LSI. TABLE II presents the comparison results on 6 OSS projects while 70% threshold is applied to the tracing methods (i.e., evaluating only the top 70% of retrieved candidate links) [56]. For space reasons, we only display VSM-POS-N and LSI-POS-N in TABLE II. VSM-POS-V and LSI-POS-V have similar performances. Since we perform 10-fold cross-validation, for each round of testing, we average the results of all requirements in each project. TABLE II presents the average values of 10 testing rounds.

As shown in TABLE II, VSM-POS-N and LSI-POS-N outperform the baseines in 6 projects. However, there is no significant improvement. Our polysemy enhanced approaches significantly improve the precision and F_2, at the same time, they do not decrease the recall on 5 projects. This confirms our hypothesis that resolving polysemy will lead to ruling out incorrect links. The only outlier is DASHBUILDER project. It seems that our FNN models fail to resolve polysemy in this project. A closer analysis on DASHBUILDER project shows that the failure does not come from failing to identify polysemous terms. For instance, while retrieving the DASHBUILDE-246 [57], our enhanced approaches have reduced the similarity scores of all 14 false positives. However, since its correct link GUVNOR-3406 [58] is from other projects, and does not share any terms with it, the similarity of its correct link is 0. Therefore, even after reducing incorrect links' similarity scores, the correct link is still hard to be retrieved because of vocabulary mismatch [59].

According to TABLE II, LSI-Polysemy improves LSI precision at 0.01 level in most cases. However, VSM-Polysemy improves VSM at 0.05 level. Our FNN models are designed to resolve polysemy problem, and previous research indicates that LSI outperforms VSM by fully solving another natural

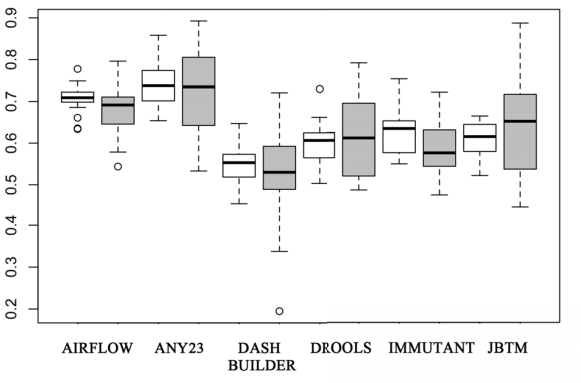

Fig. 3. F_2 differences between including all feature groups (white) and removing stakeholder features (gray) on six OSS projects: VSM-Polysemy

language (NL) problem: synonymy [22] (multiple terms have the same meaning). Therefore, we can conclude that methods of solving polysemy and synonymy are complementary with each other, i.e., the improvement achieved by resolving both of them is greater than the sum of improvements achieved by resolving them separately [60].

In addition, solving the polysemy problem leads to reducing similarity scores between requirements and incorrect links. At the same time, solving synonymy problem leads to increasing similarity scores between requirements and correct links. The combination of these two effects certainly further improves the approach's accuracy. The analysis on the results shows that, there are 128, 34, 2, 51, 47, and 357 hard-to-eliminate incorrect links (dark gray part in Fig. 1) that have been removed from retrieval results of AIRFLOW, ANY23, DASH-BUILDER, DROOLS, IMMUTANT, and JBTM respectively by using LSI-Polysemy.

For IR-based approaches, we want to get some amount of recall while tolerating only a certain percentage of false positives. F-Measure is a single measure that trades off precision versus recall. Therefore, our feature ablation study focuses on comparing F_2 performances of polysemy enhanced approaches including all feature groups and removing certain feature groups. It is unsurprising that embeddings and string matching features substantially improves the approach's performance, since both of them catch terms' distributional semantic similarity [61]. Surprisingly, not like results reported in [37], stakeholder features do not significantly improve polysemy enhanced approaches. Fig. 3 visualizes the feature ablation study results of stakeholder features using VSM-Polysemy. Analysis based on LSI-Polysemy shows similar results.

We introduce the support analysis, one method of association analysis from the data mining literature [62], to test how frequently a polysemous term is used by different stakeholders to express different meanings:

$$supp(X \Rightarrow Y) = P(X,Y) = \frac{freq(X \bigcup Y)}{|T|} \qquad (11)$$

where X is a term used by two stakeholders, Y means this term expresses different meanings, and $|T|$ is the sum of occurrences of all polysemous terms. If an association has the support that is less than a user-specified value, then the association will not be considered as significant. In practice, frequency count $freq(X \bigcup Y)$ with threshold value 3 is used more often in the software engineering literature [63], [64]. Analysis of 6 OSS projects shows that, only average 12.63% terms express different meanings because of different stakeholders using them (i.e., only 12.63% of total terms satisfy $freq(X \bigcup Y) \geq 3$). We argue that, unlike general speakers in WSJ datasets used in [37], stakeholders with similar background share the terminology, and therefore tend to use the same terms consistently when describing the objects. Further analysis of the reasons causing polysemy is needed (e.g., [65], [66]), and it will help with feature selection. However, it is out of this paper's scope. According to our observations, project evolution plays a role in polysemy problem.

The bottom 2 rows in TABLE II present cross-project testing on MODIS and CM-1. FNN models trained with all 6 OSS projects are used to identify polysemous terms in these two benchmark datasets. The results show that pre-trained FNN models can significantly improve IR-based approaches on both datasets. On one hand, FNN methods are facing the problem of lacking well-defined training datasets. On the other hand, the problem of finding the appropriate parameters in neural networks models belongs to the class of NP-complete problems [67], and computational complexity of FNN training scales linearly with the size of the network [68]. If a pre-trained model could solve the same problem in different datasets, this will greatly reduce the computational cost of building FNN models. Our cross-project testing results give us the confidence to build such models.

VI. Discussion

A. Related Work

Several IR methods can be used to detect polysemous terms. Thesaurus [12] is designed to solve term mismatch problem in automated requirements tracing. A simple thesaurus T is a set of triples $\langle t_i, t_j, \alpha \rangle$, where t_i and t_j are matching thesaurus terms and α is the similarity coefficient between them. If $\langle a, b, \alpha_{ab} \rangle$ and $\langle a, c, \alpha_{ac} \rangle$ are in T, and there is no $\langle b, c, \alpha_{bc} \rangle$ or $\alpha_{bc} = 0$, we can conclude that a is a polysemous term. A complete thesaurus library can help us find polysemous terms accurately. However, manually build such complete thesaurus library is a difficult task. On the contrary, our term-pair ranking model provides an automated way to build the thesaurus library, and our cluster-pair ranking model uses such library to detect polysemous terms.

Our results show that POS tagging improves precision of the IR-based methods. However, the improvements are not significant. In fact, VSM-POS and LSI-POS can alleviate polysemy while a term's multiple meanings are classified into different linguistic categories. For instance, the term "ship" in "board ship" is a noun which means a boat, and in "ship goods" is a verb which means transport. Indexing

only nouns or verbs leads to ignoring one meaning of term "ship". Therefore, VSM-POS and LSI-POS approaches will not link them together. However, for polysemous terms whose multiple meanings are in the same linguistic category, such as "table" (e.g. "database table" and "webpage table"), VSM-POS approaches cannot distinguish their meanings. In fact, POS tagging is one of the input features of our polysemy enabled approach. The feature ablation study shows that, POS combined with other features significantly contributes to polysemous term detection.

On the other hand, ANN models have been successfully applied to support requirements traceability recovery. In order to overcome term mismatching or more specific synonymy problem, Guo *et al.* [69] proposed an approach based on recurrent neural networks (RNN) to learn semantic associations between requirements and correct links. Unlike [69], our approach improves the precision of the automated requirements tracing by resolving polysemy. Having an innovative way to combine those two methods together may further improve the accuracy of the automated requirements tracing.

B. Threats to Validity

In this paper, we proposed a ANN-based approach to tackle a typical NL problem, polysemy, in automated requirements tracing. The experimental results show that our approach can improve the precision of retrieval results. However, several factors can affect the validity our study. Construct validity is the degree to which the variables accurately measure the concepts they purport to measure [70]. In our experiment, there were minimal threats to construct validity as standard IR measures (recall, precision, and F_2), which have been widely used in requirements traceability research, were employed to assess the different methods. The feature ablation, a fundamental analysis method in machine learning, is used to provide more insights into the results. Therefore, we believe that the measures we used sufficiently capture and quantify the different aspects of automation tracing methods evaluated in this paper.

Threats to external validity impact the generalizability of results. In particular, the results of this study might not generalize beyond the underlying experimental settings [70]. One major threat to the external validity comes from the datasets used in this experiment. We mitigate this threat by testing our approach with 6 OSS projects and 2 benchmark datasets. Additionally, the sizes of projects used in the experimental are varied. Therefore scalability questions are also addressed. Another threat comes from training sets. We acknowledge that manually creating training data (i.e., polysemous terms) is an error-prone task. However, we believe having two experts worked jointly on finding polysemous terms helps mitigate this threat. In fact, inter-rater agreement analysis (Fleiss' kappa κ [33]) also demonstrated that two experts provided coherent judgments.

In addition, specific design decisions and heuristics used during the implementation can also limit the results applicability. Such decisions include the size of input and hidden layers, using ReLU as activation functions, using TFIDF weights, and training stop criteria.

C. Implications

The vast majority of requirements in practice are written in NL, and thus suffer from typical NL problems like polysemy. Therefore, our approach can be applied to solve other requirements engineering tasks, such as ambiguity [71] and requirements interaction management [72]. In addition, Wang *et al.* [73] developed a useful Eclipse plugin (the assisted tracing tool) to help improve the precision of analyst-submitted final trace matrices by allowing analysts to tag requirements and source code. Our term-pair ranking model can improve their tool by automatically finding semantically related tags of analysts' tags.

One surprising finding is that stakeholder features contributed less to identifying polysemy. Incorporating other features, like interval time between creation of two requirements, into consideration can help us find other important features. Experimental results also show that solving polysemy and synonymy together can significantly improve the accuracy of automated requirements tracing. Fully connected FNN models have been considered to be able to solve multi-class classification problems [68]. Constructing a multi-class classifier to identify both polysemy and synonymy is a promising future extension of our approach.

Answer sets of trace links in 6 projects are defined by original developers. Developers link different documents not only because those documents describe similar issues, but also because those documents belong to the same component. This means that linked documents may not share any terms. Detecting links between documents which do not share any terms is out of the scope of the VSM-based approaches (including our FNN-based approach). Therefore, having a new approach to trace links belonging to the same component can help further improve VSM-base approaches.

VII. Conclusion

Polysemy is a typical NL problem which has impacted automated requirements tracing. This paper tries to resolve polysemy by adapting NLP approaches into IR-based requirements tracing. Our contributions include: 1) quantifying polysemy's impact on automated requirements tracing; 2) proposing a semantically enhanced approach to detect polysemous terms; and 3) testing proposed approach with 6 OSS projects and 2 benchmark datasets.

Our future work includes finding and testing more features, building polysemy enhanced automated requirements tracing with other neural network models (e.g., RNN), and training a new model to resolve both polysemy and synonymy together.

Acknowledgement

The work is funded in part by the U.S. National Science Foundation (Award CCF 1350487) and the National Natural Science Foundation of China (Fund No. 61472034, 61772071, and 61690205).

REFERENCES

[1] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. the 1st IEEE International Conference on Requirements Engineering*, Colorado Springs, CO, USA, Apr. 1994, pp. 315–322.

[2] A. De Lucia, R. Oliveto, F. Zurolo, and M. D. Penta, "Improving comprehensibility of source code via traceability information: a controlled experiment," in *Proc. the 14th IEEE International Conference on Program Comprehension*, Athens, Greece, Jun. 2006, pp. 317–326.

[3] C. Arora, M. Sabetzadeh, A. Goknil, L. C. Briand, and F. Zimmer, "Change impact analysis for natural language requirements: An NLP approach," in *Proc. the 23rd IEEE International Requirements Engineering Conference*, Ottawa, ON, Canada, Aug. 2015, pp. 6–15.

[4] N. Niu, W. Wang, and A. Gupta, "Gray links in the use of requirements traceability," in *Proc. the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Seattle, WA, USA, Nov. 2016, pp. 384–395.

[5] D. Hauksdottir, A. Vermehren, and J. Savolainen, "Requirements reuse at Danfoss," in *Proc. 20th IEEE International Requirements Engineering Conference*, Chicago, IL, USA, Sep. 2012, pp. 309–314.

[6] N. Niu, J. Savolainen, Z. Niu, M. Jin, and J.-R. C. Cheng, "A systems approach to product line requirements reuse," in *IEEE Systems Journal*, vol. 8, no. 3, 2014, pp. 827–836.

[7] M. Goodrum, J. Cleland-Huang, R. R. Lutz, J. Cheng, and R. A. Metoyer, "What requirements knowledge do developers need to manage change in safety-critical systems?" in *Proc. the 25th IEEE International Requirements Engineering Conference*, Lisbon, Portugal, Sep. 2017, pp. 90–99.

[8] N. Niu, T. Bhowmik, H. Liu, and Z. Niu, "Traceability-enabled refactoring for managing just-in-time requirements," in *Proc. the 22nd IEEE International Requirements Engineering Conference*, Karlskrona, Sweden, Sep. 2014, pp. 133–142.

[9] S. Hesari, R. Behjati, and T. Yue, "Towards a systematic requirement-based test generation framework: Industrial challenges and needs," in *Proc. the 31st IEEE International Requirements Engineering Conference*, Rio de Janeiro, Brazil, Jul. 2013, pp. 261–266.

[10] C. Ghezzi, C. Menghi, A. M. Sharifloo, and P. Spoletini, "On requirements verification for model refinements," in *Proc. the 31st IEEE International Requirements Engineering Conference*, Rio de Janeiro, Brazil, Jul. 2013, pp. 62–71.

[11] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," in *IEEE Transactions on Software Engineering*, vol. 32, no. 1, 2006, pp. 4–19.

[12] J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," in *Proc. the 11th IEEE International Conference on Requirements Engineering*, Monterey Bay, CA, USA, Sep. 2003, pp. 138–147.

[13] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou, "Poirot: A distributed tool supporting enterprise-wide automated traceability," in *Proc. the 14th IEEE International Conference on Requirements Engineering*, Minneapolis/St.Paul, MN, USA, Sep. 2006, pp. 356–357.

[14] H. Kuang, J. Nie, H. Hu, P. Rempel, J. Lu, A. Egyed, and P. Mäder, "Analyzing closeness of code dependencies for improving IR-based traceability recovery," in *Proc. the 24th IEEE International Conference on Software Analysis, Evolution and Reengineering*, Klagenfurt, Austria, Feb. 2017, pp. 68–78.

[15] W. Wang, A. Gupta, N. Niu, L. D. Xu, J.-R. C. Cheng, and Z. Niu, "Automatically tracing dependability requirements via term-based relevance feedback," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, 2018, pp. 342–349.

[16] E. Knauss, D. Damian, G. Poo-Caamaño, and J. Cleland-Huang, "Detecting and classifying patterns of requirements clarifications," in *Proc. the 20th IEEE International Requirements Engineering Conference*, Chicago, IL, USA, Sep. 2012, pp. 251–260.

[17] J. H. Hayes and A. Dekhtyar, "A framework for comparing requirements tracing experiments," in *International Journal of Software Engineering and Knowledge*, vol. 15, no. 5, 2005, pp. 751–782.

[18] T. Jenkinson, "Thoroughly document how to recover a local JTA object store from a different node," 2013. [Online]. Available: https://issues.jboss.org/browse/JBTM-1644

[19] G. Trikleris, "Remove RTS subsystem's dependency on default Undertow server and host," 2013. [Online]. Available: https://issues.jboss.org/browse/JBTM-1706

[20] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," in *Requirements Engineering*, vol. 20, no. 3, 2015, pp. 281–300.

[21] B. Rosario, "Latent semantic indexing: An overview," in *Final paper INFOSYS 240. University of Berkeley*, 2000.

[22] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," in *Journal of the American Society for Information Science*, vol. 41, no. 6, 1990, pp. 391–407.

[23] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY: Cambridge University Press, 2008.

[24] J. I. Maletic and A. Marcus, "Using latent semantic analysis to identify similarities in source code to support program understanding," in *Proc. the 12th IEEE International Conference on Tools with Artificial Intelligence*, Vancouver, BC, Canada, Nov. 2000, pp. 46–53.

[25] A. Marcus and J. I. Maletic, "Identification of high-level concept clones in source code," in *Proc. the 16th IEEE International Conference on Automated Software Engineering*, San Diego, CA, USA, Nov. 2001, pp. 107–114.

[26] ——, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. the 25th International Conference on Software Engineering*, Portland, OR, USA, May 2003, pp. 125–137.

[27] A. Chowdhury and M. C. McCabe, "Improving information retrieval systems using part of speech tagging," Institute for Systems Research, MD, Tech. Rep. TR-98-48, 1987.

[28] N. Niu and S. M. Easterbrook, "Extracting and modeling product line functional requirements," in *Proc. the 16th IEEE International Requirements Engineering Conference*, Barcelona, Spain, Sep. 2008, pp. 155–164.

[29] J.-R. Falleri, M. Huchard, M. Lafourcade, C. Nebut, V. Prince, and M. Dao, "Automatic extraction of a wordnet-like identifier network from software," in *Proc. the 18th IEEE International Conference on Program Comprehension*, Braga, Portugal, Jun. 2010, pp. 4–13.

[30] A. Mahmoud and N. Niu, "Using semantics-enabled information retrieval in requirements tracing: An ongoing experimental investigation," in *Proc. the 34th Annual IEEE International Computer Software and Applications Conference*, Seoul, Korea, Jul. 2010, pp. 246–247.

[31] J. Settimi, J. Cleland-Huang, O. B. Khadra, J. Mody, W. Lukasik, and C. DePalma, "Supporting software evolution through dynamically retrieving traces to uml artifacts," in *Proc. the 7th International Workshop on Principles of Software Evolution*, Kyoto, Japan, Sep. 2004, pp. 49–54.

[32] T. Bhowmik, N. Niu, J. Savolainen, and A. Mahmoud, "Leveraging topic modeling and part-of-speech tagging to support combinational creativity in requirements engineering," in *Requirements Engineering*, vol. 20, no. 3, 2015, pp. 253–280.

[33] J. L. Fleiss and J. Cohen, "The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability," in *Educational and Psychological Measurement*, vol. 33, no. 3, 1973, pp. 613–619.

[34] K. L. Gwet, *Handbook of Inter-Rater Reliability*. Gaithersburg, MD: Advanced Analytics, 2014.

[35] J. Zheng, W. W. Chapman, R. S. Crowley, and G. K. Savova, "Coreference resolution: A review of general methodologies and applications in the clinical domain," in *Journal of Biomedical Informatics*, vol. 44, no. 6, 2011, pp. 1113–1122.

[36] N. S. Moosavi and M. Strube, "Lexical features in coreference resolution: To be used with caution," in *Proc. the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, Jul. 2017, pp. 14–19.

[37] K. Clark and C. D. Manning, "Improving coreference resolution by learning entty-level distributed representations," in *Proc. the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, Aug. 2016, pp. 643–653.

[38] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," in *Neural Networks*, vol. 2, no. 5, 1989, pp. 359–366.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[40] M. Iyyer, V. Manjunatha, J. L. Boyd-Graber, and H. D. III, "Deep unordered composition rivals syntactic methods for text classification,"

in *Proc. the 53rd Annual Meeting of the Association for Computational Linguistics*, Beijing, China, Jul. 2015, pp. 1681–1691.

[41] G. Durrett and D. Klein, "Easy victories and uphill battles in coreference resolution," in *Proc. the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, WA, USA, Oct. 2013, pp. 1971–1982.

[42] S. Martschat and M. Strube, "Latent structures for coreference resolution," in *Transactions of the Association for Computational Linguistics*, vol. 3, 2015, pp. 405–418.

[43] V. K. Ojha, A. Abraham, and V. Sásel:, "Metaheuristic design of feedforward neural networks: A review of two decades of research," in *Engineering Applications of Artificial Intelligence*, vol. 60, 2017, pp. 97–116.

[44] G. P. Zhang, "Neural networks for classification: a survey," in *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 30, no. 4, 2000, pp. 451–462.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," in *Nature*, vol. 323, 1986, pp. 533–536.

[46] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd Edition*. Upper Saddle Revier, NJ: Pearson Education, 2014.

[47] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. the 27th Annual Conference on Neural Information Processing Systems*, Lake Tahoe, NV, USA, Dec. 2013, pp. 3111–3119.

[48] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proc. the 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, MD, USA, Jun. 2014, pp. 55–60.

[49] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. the 27th International Conference on Machine Learning*, Haifa, Israel, Jun. 2010, pp. 807–814.

[50] G. Cybenko, "Approximation by superpositions of a sigmoidal function," in *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, 1989, pp. 303–314.

[51] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. the 14th International Conference on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, USA, Apr. 2011, pp. 315–323.

[52] S. Wiseman, A. M. Rush, S. M. Shieber, and J. Weston, "Learning anaphoricity and antecedent ranking features for coreference resolution," in *Proc. the 53rd Annual Meeting of the Association for Computational Linguistics*, Beijing, China, Jul. 2015, pp. 1416–1426.

[53] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proc. the 27th International Conference on Machine Learning*, Haifa, Israel, Jun. 2010, pp. 111–118.

[54] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. D. Manning, "A multi-pass sieve for coreference resolution," in *Proc. the 2010 Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, USA, Oct. 2010, pp. 402–501.

[55] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases." School of Information Technology and Engineering, University of Ottawa, Canada, 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[56] A. Mahmoud and N. Niu, "Source code indexing for automated tracing," in *Proc. the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, Honolulu, HI, USA, May 2011, pp. 3–9.

[57] D. Gutierrez, "Dynamic navigation root for runtime perspectives," 2017. [Online]. Available: https://issues.jboss.org/browse/DASHBUILDE-246

[58] J. Hrcek, "MegaMenu: improve placement of custom perspectives," 2017. [Online]. Available: https://issues.jboss.org/browse/GUVNOR-3406

[59] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in *Proc. the 25th IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Belgium, Sep. 2010, pp. 245–254.

[60] P. Milgroma and J. Robertsb, "Complementarities and fit strategy, structure, and organizational change in manufacturing," in *Journal of Accounting and Economics*, vol. 19, no. 2-3, 1995, pp. 179–208.

[61] Y. Li, L. Xu, F. Tian, L. Jiang, X. Zhong, and E. Chen, "Word embedding revisited: A new representation learning and explicit matrix factorization perspective," in *Proc. the 24th International Joint Conference on Artificial Intelligence*, Buenos Aires, Argentina, Jul. 2015, pp. 3650–3656.

[62] J. Han, M. Kambe, and J. Pei, *Data Mining: Concepts and Techniques*. Waltham, MA: Morgan Kaufmann, 2011.

[63] A. Michail, "Data mining library reuse patterns in user-selected applications," in *Proc. the 14th IEEE International Conference on Automated Software Engineering*, Cocoa Beach, FL, USA, Oct. 1999, pp. 24–33.

[64] T. Zimmermann, P. Weiβgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *IEEE Transactions on Software Engineering*, vol. 31, no. 6, 2005, pp. 429–445.

[65] N. Niu and S. M. Easterbrook, "Managing terminological interference in goal models with repertory grid," in *Proc. the 14th IEEE International International Conference on Requirements Engineering*, Minneapolis/St.Paul, Minnesota, USA, Sep. 2006, pp. 296–299.

[66] N. Niu and S. Easterbrook, "So, you think you know others' goals? a repertory grid study," in *IEEE Software*, vol. 24, no. 2, 2007, pp. 53–61.

[67] R. Rojas, *Neural Networks*. Berlin, Germany: Springer-Verlag, 1996.

[68] Y. Goldberg, "A primer on neural network models for natural language processing," in *Journal of Artificial Intelligence Research*, vol. 57, 2016, pp. 345–420.

[69] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *Proc. the 39th International Conference on Software Engineering*, Buenos Aires, Argentina, May 2017, pp. 3–14.

[70] A. Dean, D. Voss, and D. Draguljić, *Design and Analysis of Experiments, 2nd Edition*. Cham, Switzerland: Springer, 2017.

[71] H. Yang, A. N. D. Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Extending nocuous ambiguity analysis for anaphora in natural language requirements," in *Proc. the 18th IEEE International Requirements Engineering Conference*, Sydney, Australia, Sep. 2010, pp. 25–34.

[72] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements interaction management," in *ACM Computer Surveys*, vol. 35, no. 2, 2003, pp. 132–190.

[73] W. Wang, N. Niu, H. Liu, and Y. Wu, "Tagging in assisted tracing," in *Proc. the 8th IEEE/ACM International Symposium on Software and Systems Traceability*, Florence, Italy, May 2015, pp. 8–14.