Recovering Traceability Links in Requirements Documents

Zeheng Li Mingrui Chen LiGuo Huang
Department of Computer Science & Engineering
Southern Methodist University
Dallas, TX 75275-0122
{zehengl,mingruic,lghuang}@smu.edu

Vincent Ng
Human Language Technology Institute
University of Texas at Dallas
Richardson, TX 75083-0688
vince@hlt.utdallas.edu

Abstract

Software system development is guided by the evolution of requirements. In this paper, we address the task of *requirements traceability*, which is concerned with providing bi-directional traceability between various requirements, enabling users to find the origin of each requirement and track every change made to it. We propose a *knowledge-rich* approach to the task, where we extend a supervised baseline system with (1) additional training instances derived from human-provided annotator rationales; and (2) additional features derived from a hand-built ontology. Experiments demonstrate that our approach yields a relative error reduction of 11.1–19.7%.

1 Introduction

Software system development is guided by the evolution and refinement of requirements. Requirements specifications, which are mostly documented using natural language, are refined with additional design details and implementation information as the development life cycle progresses. A crucial task throughout the entire development life cycle is *requirements traceability*, which is concerned with linking requirements in which one is a *refinement* of the other.

Specifically, one is given a set of high-level (coarse-grained) requirements and a set of low-level (fine-grained) requirements, and the goal of requirements traceability is to find for each high-level requirement all the low-level requirements that refine it. Note that the resulting mapping between high- and low-level requirements is *many-to-many*, because a low-level requirement can potentially refine more than one high-level requirement. As an example, consider the three high-level requirements and two low-level requirements shown in Figure 1 about the well-known Pine email system. In this example, three traceability links should be established: (1) HR01 is refined by UC01 (because UC01 specifies the shortcut key for saving an entry in the address book); (2) HR02 is refined by UC01 (because UC01 specifies how to store contacts in the address book); and (3) HR03 is refined by UC02 (because both of them are concerned with the help system).

From a text mining perspective, requirements traceability is a very challenging task. First, there could be abundant information irrelevant to the establishment of a link in one or both of the requirements. For instance, all the information under the Description section in UC01 is irrelevant to the establishment of the link between UC01 and HR02. Worse still, as the goal is to induce a many-to-many mapping, information irrelevant to the establishment of one link could be relevant to the establishment of another link involving the same requirement. For instance, while the Description section is irrelevant when linking UC01 and HR02, it is crucial for linking UC01 and HR01. Above all, a link can exist between a pair of requirements (HR01 and UC01) even if they do not possess any overlapping or semantically similar content words.

Virtually all existing approaches to the requirements traceability task were developed in the software engineering (SE) research community. Related work on this task can be broadly divided into two categories. In *manual approaches*, requirements traceability links are recovered manually by developers. *Automated approaches*, on the other

Proceedings of the 19th Conference on Computational Language Learning, pages 237–246,
Beijing, China, July 30-31, 2015.

High-Level Requirements

HR01
The underlined character in each menu selection shall be a shortcut key. When control and the shortcut key are pressed, the menu selection should be loaded.

HR02
The system shall have an address book available to store contacts.

HR03
The system shall have a help system that offers tips and explanation for each screen and each item on the screens upon demand.

……

Low-Level Requirements

UC01	
Use case name:	store a contact's information
Summary:	the address book should store a contact's name, email, address and phone number
Description:	1. enter "pine" command in terminal 2. either enter "a" or use arrows to make "address book" line highlighted and enter "enter" 3. enter "@" 4. enter nickname, fullname, fcc, comment and addresses. may leave some fields blank 5. press ctrl+x to save the entry

UC02	
Use case name:	access help system
Summary:	user accesses help system
Description:	user presses help key

……

Figure 1: Samples of high- and low-level requirements.

hand, have relied on information retrieval (IR) techniques, which recover links based on computing the similarity between a given pair of requirements. Hence, such similarity-based approaches are unable to recover links between those pairs that do not contain overlapping or semantically similar words or phrases.

In light of this weakness, we recast requirements traceability as a supervised binary classification task, where we classify each pair of high- and low-level requirements as positive (having a link) or negative (not having a link). In particular, we propose a *knowledge-rich* approach to the task, where we extend a supervised baseline employing only word pairs and LDA-induced topics as features (see Section 4) with two types of human-supplied knowledge. First, we employ *annotator rationales*. In the context of requirements traceability, rationales are human-annotated words or phrases in a pair of high- and low-level requirements that motivated a human annotator to establish a link between the two. In other words, rationales contain the information relevant to the establishment of a link. Therefore, using them could allow a learner to focus on the relevant portions of a requirement. Motivated by Zaidan et al. (2007), we employ rationales to create additional training instances for the learner.

Second, we employ an ontology hand-built by a domain expert. A sample ontology built for the Pine domain is shown in Table 1. As we can see, the ontology contains a *verb clustering* and a *noun clustering*: the verbs are clustered by the function they perform, whereas a noun cluster corresponds to a (domain-specific) semantic type. We employ the ontology to derive additional features.

There are at least two reasons why the ontology-based features might be useful for identifying traceability links. First, since only those verbs and nouns that (1) appear in the training data and (2) are deemed relevant by the domain expert for link identification are included in the ontology, it provides guidance to the learner as to which words/phrases in the requirements it should focus on in the learning process.[1] Second, the verb and noun clusters provide a robust generalization of the words/phrases in the requirements. For instance, a word pair that is relevant for link identification may still be ignored by the learner due to its infrequency of occurrence. The features computed based on these clusters, on the other hand, will be more robust to the infrequency problem and could therefore provide better generalizations.

Our contributions are three-fold. First, the knowledge-rich approach we propose for requirements traceability significantly outperforms a supervised baseline on two traceability datasets, Pine and WorldVistA. Second, we increase the NLP community's awareness of this under-studied, challenging, yet important problem in SE, which could lead to fruitful inter-disciplinary collaboration. Third, to facilitate future research on this problem, we make our annotated resources, including the datasets, the rationales, and the ontolo-

[1] Note that both the rationales and the words/phrases in the ontology could help the learner by allowing it to focus on *relevant* materials in a given pair of requirements. Nevertheless, they are not identical: rationales are words/phrases that are relevant to the establishment of a particular traceability link, whereas the words/phrases in the ontology are relevant to link establishment in general in the given domain.

Category	Terms
Message	mail, message, email, e-mail, PDL, subjects
Contact	contact, addresses, multiple addresses
Folder	folder, folder list, tree structure
Location	address book, address field, entry, address
Platform	windows,unix,window system,unix system
Module	help system, spelling check, Pico, shell
Protocol	MIME,SMTP
Command	shortcut key, ctrl+c, ctrl+m, ctrl+p, ctrl+x

(a) Noun clustering

Category	Terms
System Operation	evoke, operate, set up, activate, log
Message Search	search, find
Contact Manipulation	add, store, capture
Message Manipulation	compose, delete, edit, save, print
Folder Manipulation	create, rename, delete, nest
Message Communication	reply, send, receive, forward, cc, bcc
User Input	input, type, enter, press, hit, choose
Visualization	display, list, show, prompt, highlight
Movement	move,navigate
Function	support, have, perform, allow, use

(b) Verb clustering

Table 1: Manual ontology for Pine.

gies, publicly available.[2]

2 Related Work

Related work on traceability link prediction can be broadly divided into two categories, manual approaches and automatic approaches.

Manual requirement tracing. Traditional manual requirements tracing is usually accomplished by system analysts with the help of requirement management tools, where analysts visually examine each pair of requirements documented in the requirement management tools to build the Requirement Traceability Matrix (RTM). Most existing requirement management tools (e.g., Rational DOORS[3], Rational RequisitePro[4], CASE[5]) support traceability analysis. Manual tracing is often based on observing the potential relevance between a pair of requirements belonging to different categories or at different levels of detail. The manual process is human-intensive and error-prone given a large set of requirements.

[2]See our website at `http://lyle.smu.edu/~lghuang/research/Traceability/` for these annotated resources.

[3]`http://www-03.ibm.com/software/products/en/ratidoor`

[4]`http://www.ibm.com/developerworks/downloads/r/rrp`

[5]`http://www.analysttool.com`

Automated requirement tracing. Automated or semi-automated requirements traceability, on the other hand, generates traceability links automatically, and hence significantly increases efficiency. Pierce (1978) designed a tool that maintains a requirements database to aid automated requirements tracing. Jackson (1991) proposed a keyphrase-based approach for tracing a large number of requirements of a large Surface Ship Command System. More advanced approaches relying on information retrieval (IR) techniques, such as the *tf-idf*-based vector space model (Sundaram et al., 2005), Latent Semantic Indexing (Lormans and Van Deursen, 2006; De Lucia et al., 2007; De Lucia et al., 2009), probabilistic networks (Cleland-Huang et al., 2005), and Latent Dirichlet Allocation (Port et al., 2011), have been investigated, where traceability links were generated by calculating the textual similarity between requirements using similarity measures such as Dice, Jaccard, and Cosine coefficients (Dag et al., 2002). All these methods were developed based on either matching keywords or identifying similar words across a pair of requirements, and none of them have studied the feasibility of employing supervised learning to accomplish this task, unlike ours.

3 Datasets

For evaluation, we employ two publicly-available datasets annotated with traceability links. The first dataset, annotated by Sultanov and Hayes (2010), involves the Pine email system developed at the University of Washington. The second dataset, annotated by Cleland-Huang et al. (2010), involves WorldVistA, an electronic health information system developed by the USA Veterans Administration. Statistics on these datasets are shown in Table 2. For Pine, 2499 instances can be created by pairing the 49 high-level requirements with the 51 low-level use cases. For WorldVistA, 9193 instances can be created by pairing the 29 high-level requirements with the 317 low-level specifications. As expected, these datasets have skewed class distributions: only 10% (Pine) and 4.3% (WorldVistA) of the pairs are linked.

While these datasets have been annotated with traceability links, they are not annotated with annotator rationales. Consequently, we employed a software engineer specializing in requirements traceability to perform rationale annotation. We asked him to annotate rationales for a pair of re-

Datasets	Pine	WorldVistA
# of (high-level) requirements	49	29
# of (low-level) specifications	51	317
Avg. # of words per requirement	17	18
Avg. # of words per specification	148	26
Avg. # of links per requirement	5.1	13.6
Avg. # of links per specification	4.9	1.2
# of pairs that have links	250	394
# of pairs that do not have links	2249	8799

Table 2: Statistics on the datasets.

quirements only if he believed that there should be a traceability link between them. The reason is that in traceability prediction, the absence of a traceability link between two requirements is attributed to the *lack* of evidence that they should be linked, rather than the presence of evidence that they should not be linked. More specifically, we asked the annotator to identify as rationales all the words/phrases in a pair of requirements that motivated him to label the pair as positive. For instance, for the link between HR01 and UC01 in Figure 1, he identified two rationales from HR01 ("shortcut key" and "control and the shortcut key are pressed") and one from UC01 ("press ctrl+x").

4 Hand-Building the Ontologies

A traceability link prediction ontology is composed of a verb clustering and a noun clustering. We asked a software engineer with expertise in requirements traceability to hand-build the ontology for each of the two datasets. Using his domain expertise, the engineer first identified the noun categories and verb categories that are relevant for traceability prediction. Then, by inspecting the training data, he manually populated each noun/verb category with words and phrases collected from the training data.

As will be discussed in Section 8, we evaluate our approach using five-fold cross validation. Since the nouns/verbs in the ontology were collected only from the training data, the software engineer built five ontologies for each dataset, one for each fold experiment. Hence, nouns/verbs that appear in only the test data in a fold experiment are *not* included in that experiment's ontology. In other words, our test data are truly held-out w.r.t. the construction of the ontology. Tables 1 and 3 show the complete lists of noun and verb categories identified for Pine and WorldVistA, respectively, as well as sample nouns and verbs that populate each category. Note that the five ontologies employ the *same* set of noun and verb categories,

Category	Terms
Signature	signature, co-signature, identity
Prescription	prescription, electronic prescription
Authorization	authorized users, administrator
Patient Info	patient data, health summary
General Info	information, data
Component	platform, interface, integration
Laboratory	lab test results, laboratory results
Customization	individual customization,customization
Discharge	discharge, discharge instruction
Records	progress note, final note, saved note
Details	specifications, order details
Medication	medications, drug
Side-effect	allergy, drug-drug interaction, reaction
Code	ICD-9, standards, management code
User	user class hierarchy, roles, user roles
Order	orderable file, order, medication order
List	problem list,problem/diagnosis list
Rules	business rules, C32, HITSP C32
Document	documents, level 2 CCD documents
Schedule	appointments, schedule, reminders
Warning	warning, notice warning
Encounter	encounter, encounter data
Hospitalization	hospitalization data,procedure data
Arrangement	templates, clinical templates, data entry
Immunization	immunization, immunization record
Protocol	protocol,HL7,HTTP,FTP,HL7-ASTM
System data	codified data,invalid data,data elements
Key	key, security key, computer key
Identity	social security number,account number
Audit	audit log, audit records, audit trail
Duty	assignment, task

(a) Noun clustering

Category	Terms
Interface actions	click, select, search, browse
Authentication	sign, co-sign, cosign, authenticate
Customization	fit, customize, individualize
Notification	check, alert
Security control	control, prevent, prohibit, protect
Data manipulation	capture,associate,document,create
Visualization	display, view, provide, generate
Recording	audit, log
Data retrieval	export, retrieve
Deletion	remove, delete
System operation 1	save, retain
System operation 2	abort, restrict, delay, lock
Search	find, query
Communication	forward, redirect

(b) Verb clustering

Table 3: Manual ontology for WorldVistA.

differing only w.r.t. the nouns and verbs that populate each category. As we can see, for Pine, eight groups of nouns and ten groups of verbs are defined, and for WorldVistA, 31 groups of nouns and 14 groups of verbs are defined. Each noun category represents a domain-specific semantic class, and each verb category corresponds to a function performed by the action underlying a verb.

5 Baseline Systems

In this section, we describe three baseline systems for traceability prediction.

5.1 Unsupervised Baselines

The Tf-idf baseline. We employ *tf-idf* as our first unsupervised baseline. Each document is represented as a vector of unigrams. The value of each vector entry is its associated word's *tf-idf* value. Cosine similarity is used to compute the similarity between two documents. Any pair of requirements whose similarity exceeds a given threshold is labeled as positive.

The LDA baseline. We employ LDA (Blei et al., 2003) as our second unsupervised baseline. We train an LDA model on our data to produce n topics. For Pine, we set n to 10, 20, ..., 50. For WorldVistA, because of its larger size, we set n to 50, 60, ..., 100. We then represent each document as a vector of length n, with each entry set to the probability that the document belongs to one of the topics. Cosine similarity is used as the similarity measure. Any pair of requirements whose similarity exceeds a given threshold is labeled as positive.

5.2 Supervised Baseline

Each instance corresponds to a high-level requirement and a low-level requirement. Hence, we create instances by pairing each high-level requirement with each low-level requirement. The class value of an instance is positive if the two requirements involved should be linked; otherwise, it is negative. Each instance is represented using two types of features:

Word pairs. We create one binary feature for each word pair (w_i, w_j) collected from the training instances, where w_i and w_j are words appearing in a high-level requirement and a low-level requirement respectively. Its value is 1 if w_i and w_j appear in the high-level and low-level pair under consideration, respectively.

LDA-induced topic pairs. Motivated by previous work, we create features based on the topics induced by an LDA model for a requirement. Specifically, we first train an LDA model on our data to obtain n topics, where n is to be tuned jointly with C on the development (dev) set.[6] Then, we create one binary feature for each topic pair (t_i, t_j), where t_i and t_j are the topics corresponding to a high-level requirement and a low-level requirement, respectively. Its value is 1 if t_i and t_j are the most probable topics of the high-level and low-level pair under consideration, respectively.

We employ LIBSVM (Chang and Lin, 2011) to train a binary SVM classifier on the training set. We use a linear kernel, setting all learning parameters to their default values except for the C (regularization) parameter, which we tune jointly with n (the number of LDA-induced topics) to maximize F-score on the dev set.[7] Since we conduct five-fold cross validation, in all experiments that require a dev set, we use three folds for training, one fold for dev, and one fold for evaluation.

6 Exploiting Rationales

In this section, we describe our first extension to the baseline: exploiting rationales to generate additional training instances for the SVM learner.

6.1 Background

The idea of using annotator rationales to improve binary text classification was proposed by Zaidan et al. (2007). A rationale is a human-annotated text fragment that motivated an annotator to assign a particular label to a *training* document. In their work on classifying the sentiment expressed in movie reviews as positive or negative, Zaidan et al. generate additional *training* instances by removing rationales from documents. Since these pseudo-instances lack information that the annotators thought was important, an SVM learner should be less confident about the labels of these weaker instances, and should therefore place the hyperplane closer to them. A learner that successfully learns this difference in confidence assigns a higher importance to the pieces of text that are present only in the original instances. Thus the pseudo-instances help the learner both by indicating which parts of the documents are important and by increasing the number of training instances.

6.2 Application to Traceability Prediction

Unlike in sentiment analysis, where rationales can be identified for both positive and negative training reviews, in traceability prediction, rationales

[6] As in the LDA baseline, for Pine we set n to 10, 20, ..., 50, and for WorldVistA, we set n to 50, 60, ..., 100.

[7] C was selected from the set {1, 10, 100, 1000, 10000}.

can only be identified for the positive training instances (i.e., pairs with links). As noted before, the reason is that in traceability prediction, an instance is labeled as negative because of the *absence* of evidence that the two requirements involved should be linked, rather than the presence of evidence that they should not be linked.

Using these rationales, we can create *positive* pseudo-instances. Note, however, that we *cannot* employ Zaidan et al.'s method to create positive pseudo-instances. According to their method, we would (1) take a pair of linked requirements, (2) remove the rationales from both of them, (3) create a positive pseudo-instance from the remaining text fragments, and (4) add a constraint to the SVM learner forcing it to classify it less confidently than the original positive instance. Creating positive pseudo-instances in this way is problematic for our task. The reason is simple: as discussed previously, a negative instance in our task stems from the *absence* of evidence that the two requirements should be linked. In other words, after removing the rationales from a pair of linked requirements, the pseudo-instance created from the remaining text fragments should be labeled as negative.

Given this observation, we create a *positive* pseudo-instance from each pair of linked requirements by removing any text fragments from the pair that are *not* part of a rationale. In other words, we use *only* the rationales to create positive pseudo-instances. This has the effect of amplifying the information present in the rationales.

As mentioned above, while Zaidan et al.'s method cannot be used to create positive pseudo-instances, it can be used to create negative pseudo-instances. For each pair of linked requirements, we create three negative pseudo-instances. The first one is created by removing all and only the rationales from the high-level requirement in the pair. The second one is created by removing all and only the rationales from the low-level requirement in the pair. The third one is created by removing all the rationales from both requirements in the pair.

To better understand our annotator rationale framework, let us define it more formally. Recall that in a standard soft-margin SVM, the goal is to find $\mathbf{w}$ and ξ to minimize

$$\frac{1}{2}|\mathbf{w}|^2 + C\sum_i \xi_i$$

subject to

$$\forall i : c_i \mathbf{w} \cdot \mathbf{x}_i \geq 1 - \xi_i, \xi_i > 0$$

where $\mathbf{x}_i$ is a training example; $c_i \in \{-1, 1\}$ is the class label of $\mathbf{x}_i$; ξ_i is a slack variable that allows $\mathbf{x}_i$ to be misclassified if necessary; and $C > 0$ is the misclassification penalty (a.k.a. the regularization parameter).

To enable this standard soft-margin SVM to also learn from the positive pseudo-instances, we add the following constraints:

$$\forall i : \mathbf{w} \cdot \mathbf{v}_i \geq \mu(1 - \xi_i),$$

where $\mathbf{v}_i$ is the positive pseudo-instance created from positive example $\mathbf{x}_i$, $\xi_i \geq 0$ is the slack variable associated with $\mathbf{v}_i$, and μ is the margin size (which controls how confident the classifier is in classifying the pseudo-instances).

Similarly, to learn from the negative pseudo-instances, we add the following constraints:

$$\forall i, j : \mathbf{w} \cdot \mathbf{u}_{ij} \leq \mu(1 - \xi_{ij}),$$

where $\mathbf{u}_{ij}$ is the jth negative pseudo-instance created from positive example $\mathbf{x}_i$, $\xi_{ij} \geq 0$ is the slack variable associated with $\mathbf{u}_{ij}$, and μ is the margin size.

We let the learner decide how confidently it wants to classify these additional training instances based on the dev data. Specifically, we tune this *confidence* parameter μ jointly with the C value to maximize F-score on the dev set.[8]

7 Extension 2: Exploiting an Ontology

Next, we describe our second extension to the baseline: exploiting ontology-based features.

7.1 Ontology-Based Features

As mentioned before, we derive additional features for the SVM learner from the verb and noun clusters in the hand-built ontology. Specifically, we derive five types of features:

Verb pairs. We create one binary feature for each verb pair (v_i, v_j) collected from the training instances, where (1) v_i and v_j appear in a high-level requirement and a low-level requirement respectively, and (2) both verbs appear in the ontology. Its value is 1 if v_i and v_j appear in the high-level and low-level pair under consideration, respectively. Using these verb pairs as features may

[8] C was selected from the set {1, 10, 100, 100, 10000}, and μ was selected from the set {0.2, 0.3, 1, 3, 5}.

allow the learner to focus on verbs that are relevant to traceability prediction.

Verb group pairs. For each verb pair feature described above, we create one binary feature by replacing each verb in the pair with its cluster id in the ontology. Its value is 1 if the two verb groups in the pair appear in the high-level and low-level pair under consideration, respectively. These features may enable the resulting classifier to provide robust generalizations in cases where the learner chooses to ignore certain useful verb pairs owing to their infrequency of occurrence.

Noun pairs. We create one binary feature for each noun pair (n_i, n_j) collected from the training instances, where (1) n_i and n_j appear in a high-level requirement and a low-level requirement respectively, and (2) both nouns appear in the ontology. Its value is computed in the same manner as the verb pairs. These noun pairs may help the learner to focus on verbs that are relevant to traceability prediction.

Noun group pairs. For each noun pair feature described above, we create one binary feature by replacing each noun in the pair with its cluster id in the ontology. Its value is computed in the same manner as the verb group pairs. These features may enable the classifier to provide robust generalizations in cases where the learner chooses to ignore certain useful noun pairs owing to their infrequency of occurrence.

Dependency pairs. In some cases, the noun/verb pairs may not provide sufficient information for traceability prediction. For example, the verb pair feature (*delete*, *delete*) is suggestive of a positive instance, but the instance may turn out to be negative if one requirement concerns deleting messages and the other concerns deleting folders. As another example, the noun pair feature (*folder*, *folder*) is suggestive of a positive instance, but the instance may turn out to be negative if one requirement concerns creating folders and the other concerns deleting folders.

In other words, we need to develop features that encode the relationship between verbs and nouns. To do so, we first parse each requirement using the Stanford dependency parser (de Marneffe et al., 2006), and collect each noun-verb pair (n_i,v_j) connected by a dependency relation. We then create binary features by pairing each related noun-verb pair found in a high-level training requirement with each related noun-verb pair found in a low-level training requirement. The feature value is 1 if the two noun-verb pairs appear in the pair of requirements under consideration. To enable the learner to focus on learning from relevant verbs and nouns, only verbs and nouns that appear in the ontology are used to create these features.

7.2 Learning the Ontology

An interesting question is: is it possible to learn an ontology rather than hand-building it? This question is of practical relevance, as hand-constructing the ontology is a time-consuming and error-prone process. Below we describe the steps we propose for ontology learning.

Step 1: Verb/Noun selection. We select the nouns, noun phrases (NPs) and verbs in the training set to be clustered. Specifically, we select a verb/noun/NP if (1) it appears more than once in the training data; (2) it contains at least three characters (thus avoiding verbs such as *be*); and (3) it appears in the high-level but not the low-level requirements and vice versa.

Step 2: Verb/Noun representation. We represent each noun/NP/verb as a feature vector. Each verb v is represented using the set of nouns/NPs collected in Step 1. The value of each feature is binary: 1 if the corresponding noun/NP occurs as the direct or indirect object of v in the training data (as determined by the Stanford dependency parser), and 0 otherwise. Similarly, each noun n is represented using the set of verbs collected in Step 1. The value of each feature is binary: 1 if n serves as the direct or indirect object of the corresponding verb in the training data, and 0 otherwise.

Step 3: Clustering. To produce a verb clustering and a noun clustering, we cluster the verbs and the nouns/NPs separately using the single-link algorithm. Single-link is an agglomerative algorithm where each object to be clustered is initially in its own cluster. In each iteration, it merges the two most similar clusters and stops when the desired number of clusters is reached. Since we are using single-link clustering, the similarity between two clusters is the similarity between the two most similar objects in the two clusters. We compute the similarity between two objects by taking the dot product of their feature vectors.

Since we do not know the number of clusters to be produced *a priori*, for Pine we produce three noun clusterings and three verb clusterings (with 10, 15, and 20 clusters each). For WorldVistA,

given its larger size, we produce five noun clusterings and five verb clusterings (with 10, 20, 30, 40, and 50 clusters each). We then select the combination of noun clustering, verb clustering, and C value that maximizes F-score on the dev set, and apply the resulting combination on the test set.

To compare the usefulness of the hand-built and induced ontologies, in our evaluation we will perform separate experiments in which each ontology is used to derive the features from Section 7.1.

8 Evaluation

8.1 Experimental Setup

We employ as our evaluation measure F-score, which is the unweighted harmonic mean of recall and precision. Recall (R) is the percentage of links in the gold standard that are recovered by our system. Precision (P) is the percentage of links recovered by our system that are correct. We preprocess each document by removing stopwords and stemming the remaining words. All results are obtained via five-fold cross validation.

8.2 Results and Discussion

Results on Pine and WorldVistA are shown in Table 4(a) and Table 4(b), respectively.

8.2.1 No Pseudo-instances

The "No pseudo" column of Table 4 shows the results when the learner learns from only real training instances (i.e., no pseudo-instances). Specifically, rows 1 and 2 show the results of the two unsupervised baselines, *tf-idf* and LDA, respectively.

Recall from Section 5.1 that in both baselines, we compute the cosine similarity between a pair of requirements, positing them as having a traceability link if and only if their similarity score exceeds a threshold that is tuned based on the *test* set. By doing so, we are essentially giving both unsupervised baselines an unfair advantage in the evaluation. As we can see from rows 1 and 2 of the table, *tf-idf* achieves F-scores of 54.5% on Pine and 46.5% on WorldVistA. LDA performs significantly worse than *tf-idf*, achieving F-scores of 34.2% on Pine and 15.1% on WorldVistA.[9]

Row 3 shows the results of the supervised baseline described in Section 5.2. As we can see, this baseline achieves F-scores of 57.5% on Pine and 63.3% on WorldVistA, significantly outperforming the better unsupervised baseline (*tf-idf*) on both datasets. When this baseline is augmented with features derived from manual clusters (row 4), the resulting system achieves F-scores of 62.6% on Pine and 64.2% on WorldVistA, outperforming the supervised baseline by 5.1% and 0.9% in F-score on these datasets. These results represent significant improvements over the supervised baseline on both datasets, suggesting the usefulness of the features derived from manual clusters for traceability link prediction. When employing features derived from induced rather than manual clusters (row 5), the resulting system achieves F-scores of 61.7% on Pine and 64.6% on WorldVistA, outperforming the supervised baseline by 4.2% and 1.3% in F-score on these datasets. These results also represent significant improvements over the supervised baseline on both datasets. In addition, the results obtained using manual clusters (row 4) and induced clusters (row 5) are statistically indistinguishable. This result suggests that the ontologies we induced can potentially be used in lieu of the manually constructed ontologies for traceability link prediction.

8.2.2 Using Positive Pseudo-instances

The "Pseudo pos only" column of Table 4 shows the results when each of the systems is trained with additional positive pseudo-instances.

Comparing the first two columns, we can see that employing positive pseudo-instances increases performance on Pine (F-scores rise by 0.7–1.1%) but decreases performance on WorldVistA (F-scores drop by 0.3–2.1%). Nevertheless, the corresponding F-scores in all but one case (Pine, induced) are statistically indistinguishable. These results seem to suggest that the addition of positive pseudo-instances is not useful for traceability link prediction.

Note that the addition of features derived from manual/induced clusters to the supervised baseline no longer consistently improves its performance: while F-scores still rise significantly by 4.6–5.5% on Pine, they drop insignificantly by 0.1–0.5% on WorldVistA.

8.2.3 Using Positive and Negative Pseudo-instances

The "Pseudo pos+neg" column of Table 4 shows the results when each of the systems is trained with additional positive *and* negative pseudo-instances.

Comparing these results with the corresponding "Pseudo pos only" results, we can see that

[9] All significance tests are paired t-tests ($p < 0.05$).

	System	No pseudo R	P	F	Pseudo pos only R	P	F	Pseudo pos+neg R	P	F	Pseudo residual R	P	F
1	*Tf-idf* baseline	73.6	43.3	54.5	–	–	–	–	–	–	–	–	–
2	LDA baseline	30.4	39.2	34.2	–	–	–	–	–	–	–	–	–
3	Supervised baseline	50.4	67.0	57.5	51.2	67.3	58.2	53.9	73.8	62.3	31.6	68.6	43.2
4	+ manual clusters	54.4	73.9	62.6	55.6	74.7	63.7	57.6	77.0	65.9	30.0	72.1	42.3
5	+ induced clusters	53.6	72.8	61.7	54.8	73.6	62.8	55.2	75.0	63.6	30.0	73.5	42.6

(a) Pine

	System	No pseudo R	P	F	Pseudo pos only R	P	F	Pseudo pos+neg R	P	F	Pseudo residual R	P	F
1	*Tf-idf* baseline	60.4	37.8	46.5	–	–	–	–	–	–	–	–	–
2	LDA baseline	25.9	10.6	15.1	–	–	–	–	–	–	–	–	–
3	Supervised baseline	52.5	79.9	63.3	52.2	79.2	63.0	55.9	80.6	66.0	49.2	71.5	58.3
4	+ manual clusters	52.5	82.8	64.2	51.5	80.8	62.9	57.1	83.0	67.6	47.7	76.1	58.6
5	+ induced clusters	52.8	83.2	64.6	51.0	80.7	62.5	57.1	82.1	67.4	47.7	76.4	58.7

(b) WorldVistA

Table 4: Results of supervised systems on the Pine and WorldVistA datasets.

additionally employing negative pseudo-instances consistently improves performance: F-scores rise by 0.8–4.1% on Pine and 3.0–4.9% on WorldVistA. In particular, the improvements in F-score in three of the six cases (Pine/Baseline, WorldVistA/manual, WorldVistA/induced) are statistically significant. These results suggest that the additional negative pseudo-instances provide useful information for traceability link prediction.

In addition, the use of features derived from manual/induced clusters to the supervised baseline consistently improves its performance: F-scores rise significantly by 1.3–3.6% on Pine and significantly by 1.4–1.6% on WorldVistA.

Finally, the best results in our experiments are achieved when both positive and negative pseudo-instances are used in combination with manual/induced clusters: F-scores reach 63.6–65.9% on Pine and 67.4–67.6% on WorldVistA. These results translate to significant improvements in F-score over the supervised baseline by 6.1–8.4% on Pine and 4.1–4.3% on WorldVistA, or relative error reductions of 14.3–19.7% on Pine and 11.1–11.7% on WorldVistA.

8.2.4 Pseudo-instances from Residuals

Recall that Zaidan et al. (2007) created pseudo-instances from the text fragments that remain after the rationales are removed. In Section 6.3, we argued that their method of creating positive pseudo-instances for our requirements traceability task is problematic. In this subsection, we empirically verify the correctness of this claim.

Specifically, the "Pseudo residual" column of Table 4 shows the results when each of the "No pseudo" systems is additionally trained on the positive pseudo-instances created using Zaidan et al.'s method. Comparing these results with the corresponding "Pseudo pos+neg" results, we see that replacing our method of creating positive pseudo-instances with Zaidan et al.'s method causes the F-scores to drop significantly by 7.7–23.6% in all cases. In fact, comparing these results with the corresponding "No pseudo" results, we see that except for the baseline system, employing positive pseudo-instances created from Zaidan et al.'s method yields significantly worse results than not employing pseudo-instances at all. These results provide suggestive evidence for our claim.

9 Conclusion

We investigated a knowledge-rich approach to an important yet under-studied SE task that presents a lot of challenges to NLP researchers: traceability prediction. Experiments on two evaluation datasets showed that (1) in comparison to a supervised baseline, this method reduces relative error by 11.1–19.7%; and (2) results obtained using induced clusters were competitive with those obtained using manual clusters. To stimulate research on this task, we make our annotated resources publicly available.

Acknowledgments

We thank the three anonymous reviewers for their insightful comments on an earlier draft of the paper. This research was supported in part by the U.S. Department of Defense Systems Engineering Research Center (SERC) new project incubator fund RT-128 and the U.S. National Science Foundation (NSF Awards CNS-1126747 and IIS-1219142).

References

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27.

Jane Cleland-Huang, Raffaella Settimi, Chuan Duan, and Xuchang Zou. 2005. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 135–144.

Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. 2010. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (Volume 1)*, pages 155–164.

Johan Natt Dag, Björn Regnell, Pär Carlshamre, Michael Andersson, and Joachim Karlsson. 2002. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33.

Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology*, 16(4):13.

Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. 2009. Assessing IR-based traceability recovery tools through controlled experiments. *Empirical Software Engineering*, 14(1):57–92.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, pages 449–454.

Justin Jackson. 1991. A keyphrase based traceability scheme. In *IEEE Colloquium on Tools and Techniques for Maintaining Traceability During Design*, pages 2/1–2/4. IET.

Marco Lormans and Arie Van Deursen. 2006. Can LSI help reconstructing requirements traceability in design and test? In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, pages 47–56.

Robert A Pierce. 1978. A requirements tracing tool. *ACM SIGSOFT Software Engineering Notes*, 3(5):53–60.

Daniel Port, Allen Nikora, Jane Huffman Hayes, and LiGuo Huang. 2011. Text mining support for software requirements: Traceability assurance. In *Proceedings of the 44th Hawaii International Conference on System Sciences*, pages 1–11.

Hakim Sultanov and Jane Huffman Hayes. 2010. Application of swarm techniques to requirements engineering: Requirements tracing. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 211–220.

Senthil Karthikeyan Sundaram, Jane Huffman Hayes, and Alexander Dekhtyar. 2005. Baselines in requirements tracing. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–6.

Omar Zaidan, Jason Eisner, and Christine Piatko. 2007. Using "annotator rationales" to improve machine learning for text categorization. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 260–267.