TRACING REQUIREMENTS AS A PROBLEM OF MACHINE LEARNING

Zeheng Li and LiGuo Huang

Southern Methodist University, Dallas, Texas, USA

ABSTRACT

Software requirement engineering and evolution essential to software development process, which defines and elaborates what is to be built in a project. Requirements are mostly written in text and will later evolve to fine-grained and actionable artifacts with details about system configurations, technology stacks, etc. Tracing the evolution of requirements enables stakeholders to determine the origin of each requirement and understand how well the software's design reflects to its requirements. Reckoning requirements traceability is not a trivial task, a machine learning approach is used to classify traceability between various associated requirements. In particular, a 2-learner, ontology-based, pseudo-instances-enhanced *approach, where two classifiers are trained to separately exploit two types of features,* lexical *features and features derived from a hand-built ontology, is investigated for such task. The hand-built ontology is also leveraged to generate pseudo training instances to improve machine learning results. In comparison to a supervised baseline system that uses only lexical features, our approach yields a relative error reduction of 56.0%. Most interestingly, results do not deteriorate when the hand-built ontology is replaced with its automatically constructed counterpart.*

KEYWORDS

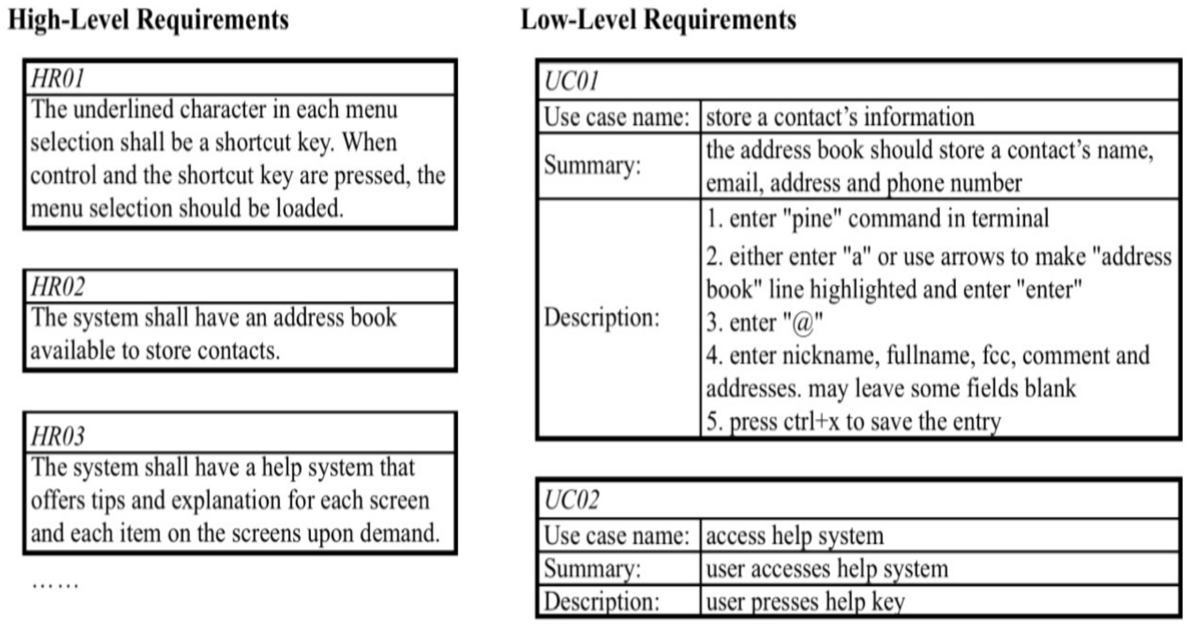
Requirements Traceability, Software Design, Machine Learning

1. INTRODUCTION

Evolution and refinement of requirements guides the software system development process by defining and specifying what to be built for a software system. Requirement specifications, mostly documented in natural language, are refined with additional design and implementation details as a software project move forwards in its development life cycle. An important task in software requirements engineering process is *requirements traceability*, which is concerned with linking requirements in which one is a *refinement* of the other. Being able to establish traceability links allows stakeholders to find the source of each requirement and track every change that has been made to it, and ensures the continuous understanding of the problem that needs to be solved so that the right system is delivered.

In practice, one is given a set of high-level (coarse-grained) requirements and a set of low-level (fine-grained) requirements, and requirements traceability aims 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 between the high-level (HR) and low-level requirements (UC): (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).



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.

From the perspective of the information retrieval and text mining, 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, the information in the Description section of UC01 appears to be 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 appears to be irrelevant to linking UC01 and UR02, a traceability linking shall exist between 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 classified into two categories: manual and automated approaches. As for manual approaches, requirements traceability links are manually recovered by developers. Automated approaches, on the other hand, have relied on information retrieval (IR) techniques, which recover links based on similarity computed 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 as mentioned above.

In light of this weakness, requirements traceability is recasted as a supervised binary classification task, where each pair of high- and low-level requirements is to be classified as positive (having a link) or negative (not having a link). Each pair of requirements is represented by two types of features. First, word pairs feature is employed, where each of which is composed of a word taken from each of the two requirements involved. These features will enable the learning algorithm to identify both semantically similar and dissimilar word pairs that are strongly indicative of a refinement relation between the two requirements, thus overcoming the aforementioned weakness associated with similarity-based approaches

Next, features are derived from an ontology hand-built by a domain expert. The sample ontology built for the Pine dataset is shown in Table 1. The ontology contains only 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.

There are at least two reasons why the ontology 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 that are computed based on these clusters, on the other hand, will be more robust to the infrequency problem and therefore potentially provide better generalizations.

Last, considering ontology a set of natural annotator rationales, pseudo training instances are used to help learners by providing indication of the importances of different parts of document as well as increasing the size of training instances.

Our main contribution in this paper lies in the proposal of a 2-learner, ontology-based, pseudo-instances-enhanced approach to the task of traceability link prediction, where, for the sake of robustness, two classifiers are trained to separately exploit the word-pair features, the ontology based features, and ontology-enhanced pseudo instances. Results on a traceability dataset involving the Pine domain reveal that our use of two learners and the ontology-based features are both key to the success of our approach: it significantly outperforms not only a supervised baseline system that uses only word pairs features, but also a system that trains a single classifier over both the word pairs and the ontology-based features. Perhaps most interestingly, results do not deteriorate when the hand-built ontology is replaced with an automatically constructed ontology. And by feeding the learners with pseudo training instances generated from ontology, the performance is improved further significantly.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 introduces the Pine dataset and our hand-built ontology is described in Section 4. Then Section 5 presents our 2-learner, ontology-based, pseudo-instances-enhanced approach to traceability link prediction. Finally, Section 6 presents evaluation results and Section 7 draws conclusions.

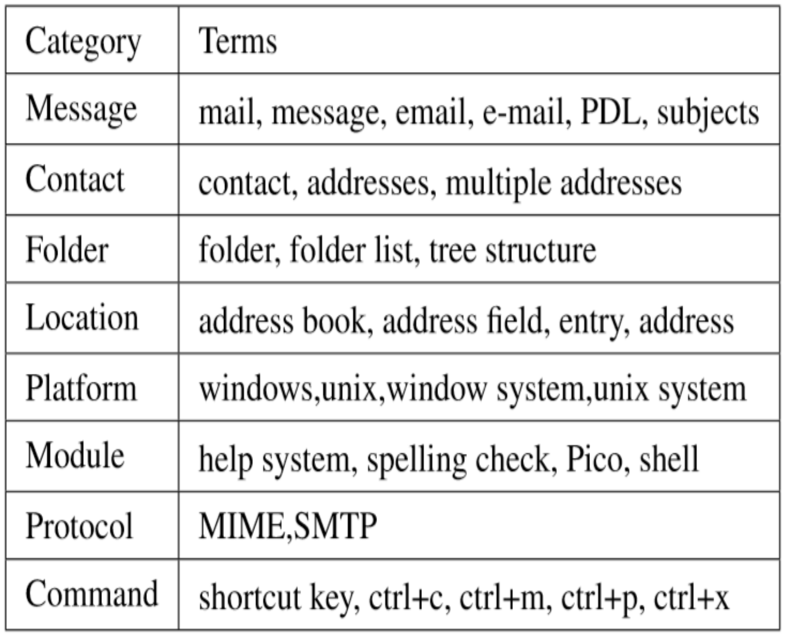
2. RELATED WORK

2.1. MANUAL APPROACHES

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, Rational Requisite-Pro, CASE) 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 details. The manual process is human-intensive and error-prone given a large set of requirements. Moreover, such domain knowledge could be lost due to requirements changes, distributed teams, or system refactoring during the life cycle of system development and evolution

Table 1. Manual ontology for Pine.

(a) Noun clustering



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

(b) Verb 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

2.2. AUTOMATED APPROACHES

Automated or semi-automated requirements traceability has been exploited by many researchers. Pierce [2] designed a tool that maintains a requirements database to aid automated requirements tracing. Jackson [3] 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 [4], Latent Semantic Indexing [5–7], probabilistic networks [8], and Latent Dirichlet Allocation [9], 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 [10]. All these methods were developed based on either matching keywords or identifying similar words across a pair of requirements. In recent years, Li [11] studied the feasibility of employing supervised learning to accomplish this task. Guo [12] applied word embedding and recurrent neural network to generate trace links.

3. DATASET

The well known Pine system is used for evaluation. This dataset consists of a set of 49 (high-level) requirements and a set of 51 (low-level) use case specifications about Pine, an email system developed at the University of Washington. Statistics on the dataset are provided on Table 2. The dataset has a skewed class distribution: out of the 2499 pairs of requirement and use case specification, only 10% (250) are considered traceability links.

Table 2. Statistics on the Pine dataset.

Number of (high-level) requirements	49
Number of (low-level) use cases	51
Avg. # of words in the requirements	17
Avg. # of words in the use cases	148
Avg. # of links per requirement	5.1
Avg. # of links per use case	4.9
# of pairs that have links	250
# of pairs that do not have links	2249

4. HAND-BUILDING THE ONTOLOGY

As mentioned before, our ontology is composed of a verb clustering and a noun clustering. A software engineer who has expertise in both requirements traceability and the Pine software domain is employed to hand-build the ontology. 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 the words and phrases collected from the training data.

As will be discussed in Section 6, our approach is evaluated by using 5-fold cross validation. Since the nouns/verbs in the ontology were collected only from the training data, the software engineer built five ontologies, one for each fold experiment. Hence, nouns/verbs that appear in only the test data in each fold experiment will *not* be in the ontology. In other words, our test data are truly held-out w.r.t. the construction of the ontology. Table 1 shows the ontology built for one of the fold experiments. Note that the five ontologies employ the *same* set of noun and verb categories, differing only w.r.t. the nouns and verbs that populate each category. As it can be seen from Table 1, eight groups of nouns and ten 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. APPROACH

This section describes our supervised approach alongside with its three extensions.

5.1 CLASSIFIER TRAINING

Each instance corresponds to a high-level requirement and a low-level requirement. Hence, instances are created 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. To conduct 5-fold cross-validation experiments, instances are randomly partitioned into five folds of roughly the same size. A classifier is trained on only four folds and evaluated on the remaining fold in each fold experiment. Each instance is represented using seven types of features, as follows

Same words. One binary feature is created for each word *w* appearing in the training data. Its value is 1 if *w* appears in *both* requirements in the pair under consideration. Hence, this feature type contains the subset of the word pair features mentioned earlier where the two words in the pair are the same.

Different words. One binary feature is created for each word pair (w_i, w_j) collected from the training instances, where w_i and w_j are non-identical 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

Verb pairs. One binary feature is created 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 allow the learner to focus on verbs that are relevant to traceabilityprediction.

Verb group pairs. For each verb pair feature described above, one binary feature is created 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. One binary feature is created 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, one binary feature is created 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, useful features are those that encode the verbs and nouns in isolation but the relationship between them. To do so, each requirement is parsed by the Stanford dependency parser [13], and each noun-verb pair (n_i ,v_j) is collected if it's connected by a dependency relation. Binary features are created 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.

LIBSVM [14] is employed as the learning algorithm for training a binary SVM classifier on the training set. In particular the linear kernel is chosen to tune the C value (the regularization parameter) to maximize F-score on the development (dev) set. All other learning parameters are set to their default values.

To improve performance, feature selection (FS) is employed by using the backward elimination algorithm [15]. Starting with all seven feature types, the algorithm iteratively removes one feature type at a time until only one feature type is left. Specifically, in each iteration, it removes the feature type whose removal yields the largest F-score on the dev set. Feature subset that achieve the largest F-score on the dev set over all iterations is picked to be applied in test set.

Note that tuning the C value (from LIBSVM) and selecting the feature subset both require the use of a dev set. In each fold experiment, one fold is reserved for development and the remaining three folds is used for training classifiers. C value is jointly tuned with the selection of feature subset that maximizes F-score on the dev set.

5.1. THREE EXTENSIONS

The following presents three extensions to our supervised approach.

5.1.1. EMPLOYING TWO VIEWS

Our first extension involves splitting our feature sets into two *views* (i.e., disjoint subsets) and training one classifier on each view. To motivate this extension, recall that the ontology is composed of words and phrases that are deemed relevant to traceability prediction according to a SE expert. In other words, the (word- and cluster-based) features derived from the ontology (i.e., features 3–7 in our feature set) are sufficient for traceability prediction, and the remaining features (features 1 and 2) are not needed according to the expert. While some of the word pairs that appear in features 1 and 2 also appear in features 3–7, most of them do not. If these expert-determined irrelevant features are indeed irrelevant, then retaining them could be harmful for classification because they significantly outnumber their relevant counterparts. However, if some of these features are relevant (because some relevant words are missed by the expert, for instance), then removing them would not be a good idea either.

Our solution to this dilemma is to divide the feature set into two views. Given the above discussion, a natural feature split would involve putting the ontology-based features (features 3–7) into one view and the remaining ones (features 1–2) into the other view. Then one SVM classifier is trained on each view as before. During test time, both classifiers are applied to a test instance, classifying it using the prediction associated with the higher confidence value.[5] This setup would prevent the expert-determined irrelevant features from affecting the relevant ones, and at the same time avoid totally discarding them in case they do contain some relevant information.

A natural question is: why not simply use backward elimination to identify the irrelevant features? While FS could help, it may not be as powerful as one would think because (1) backward elimination is greedy; and (2) the features are selected using a fairly small set of instances (i.e., the dev set) and may therefore be biased towards the dev set.

In fact, our 2-learner setup and FS are considered as *complementary* rather than *competing* solutions to our dilemma. In particular, FS is to be used in the 2-learner setup: when training the classifiers on the two views, backward elimination is employed in the same way as before by removing the feature type (from one of the two classifiers) whose removal yields the highest F-score on the dev set in each iteration.

5.1.2. LEARNING THE ONTOLOGY

An interesting question is: can the ontology be learned instead of hand-built? Not only is this question interesting from a research perspective, it is of practical relevance: even if a domain expert is available, hand-constructing the ontology is a time-consuming and error-prone process. The following describes the steps for ontology learning, which involves producing a verb clustering and a noun clustering.

Step 1: Verb/Noun selection. The nouns, noun phrases (NPs) and verbs in the training set will be clustered. Specifically, a verb/noun/NP is selected if (1) it appears more 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. Each noun/NP/verb is represented 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. Verbs and the nouns/NPs are clustered separately to produce a verb clustering and a noun clustering. Two clustering algorithms are experimented. The first one, which is referred to as *Simple*, is the classical 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. The second clustering algorithm is motivated by the following observation. A better verb clustering could be produced if each verb were represented using noun *categories* rather than nouns/NPs, because there is no need to distinguish between the nouns in the same category in order to produce the verb clusters. Similarly, a better noun clustering could be produced if each noun were represented using verb categories.

In practice, the noun and verb categories do *not* exist (because they are what the clustering algorithm is trying to produce). However, the (partial) verb clusters produced during the verb clustering process can be used to improve noun clustering and vice versa. This motivates our *Interactive* clustering algorithm. Like *Simple*, *Interactive* is also a single-link clustering algorithm. Unlike *Simple*, which produces the two clusterings separately, *Interactive* interleaves the verb and noun clustering processes, as described below.

Initially, each verb and each noun is in its own cluster. In each iteration, (1) the two most similar verb clusters is merged; (2) the noun's feature representation are updated by merging the two verb features that correspond to the newly formed verb cluster; (3) the two most similar noun clusters are merged using this updated feature representation for nouns; (4) the verb's feature representation are updated by merging the two noun features that correspond to the newly formed noun cluster. As in Simple, Interactive terminates when the desired number of clusters is reached.

For both clustering algorithms, the similarity between two objects are computed by taking the dot product of their feature vectors. Since both clustering algorithms are single-link, the similarity between two clusters is the similarity between the two most similar objects in the two clusters.

Considering the number of clusters to be produced is not known *a priori*, three noun clusterings and three verb clusterings (with 10, 15, and 20 clusters each) are generated. Then the combination of noun clustering, verb clustering, the C value, and the feature subset that maximizes F-score on the dev set is selected, and the resulting combination is applied on the test set.

5.1.3. EXPLOITING RATIONALES

This section describes another extension to the baseline: exploiting rationales to generate additional training instances for the SVM learner**.**

Background

The idea of using annotator rationales to improve text classification was proposed by Zaidan et al. [1] A rationale is a human-annotated text fragment that motivates 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 as important, an SVM learner should be less confident about the label of these weaker instances (by placing the hyperplane closer to the less confidently labeled training instances). 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 providing indication to which parts of the documents are important and by increasing the number of training instances.

Application to Traceability Prediction

Unlike in sentiment analysis, where rationales can be identified for both positive and negative training reviews, in traceability prediction, rationales 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. Hence, only *positive* pseudo-instances will be created for training a traceability predictor.

Zaidan et al.'s method *cannot* be applied *as is* to create positive pseudo-instances. According to their method, (1) a pair of linked requirements is chosen, (2) the rationales from both of them are removed, (3) a positive pseudo-instance from the remaining text fragments is created, and (4) a constraint is added to the SVM learner, forcing the learner to classify that positive pseudo-instance less confidently than the original positive instance. Creating positive pseudo-instances in this way is problematic for our task. The reason is simple: 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, one option is to employ Zaidan et al.'s method to create negative pseudo instances. Another option would be to 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, *only* the rationales are used to create positive pseudo-instances. Both options could be viable, but *positive* rather than *negative* pseudo-instances are chosen to add to our training set, as adding positive pseudo-instances will not aggravate the class imbalance problem.

Unlike Zaidan et al., who force the learner to classify pseudo-instances less confidently than the original instances, our learner decide whether it wants to classify these additional training instances more or less confidently based on the dev data. In other words, this *confidence* parameter (denoted as μ in Zaidan et al.'s paper) is tuned jointly with the C value to maximize F-score on the dev set. Note that pseudo-instances are created only for the training set, as rationales are annotated only in the training documents.

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

The following constraints are added to enable this standard soft-margin SVM to also learn from the positive pseudo-instances:

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

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 v_i, and μ is the margin size (which controls how confident the classifier is in classifying the pseudo-instances).

Similarly, the following constraints are added to learn from the negative pseudo-instances:

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

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 u_{ij}, and μ is the margin size.

Our learner decide how confidently it wants to classify these additional training instances based on the dev data. Specifically, this *confidence* parameter μ is tuned jointly with the C value to maximize F-score on the dev set.

6. EVALUATION

6.1. EXPERIMENTAL SETUP

F-score, which is the unweighted harmonic mean of recall and precision, is employed as the evaluation measure. Recall is the percentage of links in the gold standard that are recovered by our system. Precision is the percentage of links recovered by our system that are correct. Each document is preprocessed by removing stopwords and stemming the remaining words. All results are obtained via 5-fold cross validation**.**

6.2. RESULTS AND DISCUSSION

6.2.1. BASELINE SYSTEMS

There are two unsupervised and two supervised baselines.

Baseline 1: Tf.Idf. Motivated by previous work, *tf.idf* is employed as our first unsupervised baseline. Each document is represented as a vector of unigrams. The value of each feature is its *tf.idf* value. Cosine is used to compute the similarity between two documents. Any pair of requirements whose similarity exceeds a given threshold is labeled as positive. Thresholds from 0.1 to 0.9 with an increment of 0.1 are tested and results are reported using the best threshold, essentially giving an advantage to it in the performance comparison. From row 1 of Table 3, it achieves an F-score of 54.5%.

Baseline 2: LDA. Also motivated by previous work, LDA is employed as our second unsupervised baseline. An LDA is trained on our data to produce n topics (where n=10, 20, . . ., 60). Then the n topics are used as features for representing each document, where the value of a feature is the probability the document belongs to the corresponding topic. Cosine is used as the similarity measure. Any pair of requirements whose similarity exceeds a given threshold is labeled as positive. Thresholds from 0.1 to 0.9 with an increment of 0.1 are tested and results are reported using the best threshold, essentially giving an advantage to it in the performance comparison. From row 2 of Table 3, it achieves an F-score of 34.2%.

Baseline 3: Features 1 and 2. As the first supervised baseline, a SVM classifier is trained using only features 1 and 2 (all the word pairs). From row 3 of Table 3, it achieves F-scores of 57.1% (without FS) and 67.7% (with FS). These results suggest that FS is indeed useful.

Baseline 4: Features 1, 2, and LDA. As the second supervised baseline, feature set used in Baseline 3 is augmented with the LDA features used in Baseline 2 and then a SVM classifier is trained. The best n (number of topics) is selected using the dev set. Clearly from row 4 of Table 3, this is the best of the four baselines: it significantly outperforms Baseline 3 regardless of whether feature selection is performed, suggesting the usefulness of the LDA features.

6.2.2. OUR APPROACH

Next, our 2-learner, ontology-based approach is evaluated, without using pseudo instances created from rationales. In the single-learner experiments, a classifier is trained on the seven features described in Section 4.1, whereas in the 2-learner experiments, these seven features are split as described in Section 4.2.

Setting 1: Single learner, manual clusters. From row 5 of Table 3 under "No pseudo" column, this classifier significantly outperforms the best baseline (Baseline 4): F-scores increase by 4.6% (without FS) and 3.5% (with FS). Since the only difference between this and Baseline 4 lies in whether the LDA features or the ontology-based features are used, these results seem to suggest that features formed from the clusters in our hand-built ontology are more useful than the LDA feature.

Setting 2: Single learner, induced clusters. From row 6 of Table 3 under "No pseudo" column, this classifier performs statistically indistinguishably from the one in Setting 1. This is an encouraging result: it shows that even when features are created from induced rather than manual clusters, performance does not significantly drop regardless of whether FS is performed.

Setting 3: Two learners, manual clusters. From row 7 of Table 3 under "No pseudo" column, this classifier performs significantly better than the one in Setting 1: F-scores increase by 9.2% (without FS) and 3.0% (with FS). As the two settings differ only w.r.t. whether one or two learners are used, the improvements suggest the effectiveness of our 2-learner framework.

Setting 4: Two learners, induced clusters. From row 8 of Table 3 under "No pseudo" column, this classifier performs significantly better than the one in Setting 2: F-scores increase by 9.3% (without FS) and 5.8% (with FS). It also performs indistinguishably from the one in Setting 3. Taken together, these results suggest that (1) our 2-learner framework is effective in improving performance, and (2) features derived from induced clusters are as effective as those from manual clusters.

Table 3. Five-fold cross-validation results.

			No pseudo			Pseudo pos only			Pseudo pos+neg			Pseudo residual		
	System	Feature Selection?	R	P	F	R	P	F	R	P	F	R	P	F
	Baseline Systems													
1	Tf.idf	N/A	73.6	43.3	54.5	–	–	–	–	–	–	–	–	–
2	LDA	N/A	30.4	39.2	34.2	–	–	–	–	–	–	–	–	–
3	Features 1&2	No	50.0	66.5	57.1	51.0	67.2	58.0	52.4	68.2	59.3	31.1	68.2	42.7
		Yes	62.4	73.9	67.7	71.6	68.6	70.1	70.0	70.0	70.0	35.2	70.5	47.0
4	Features 1&2+LDA	No	50.4	67.0	57.5	51.2	67.3	58.2	53.9	73.8	62.3	31.6	68.6	43.2
		Yes	60.8	74.9	67.1	70.4	71.8	71.1	72.0	71.1	71.6	34.8	70.3	46.6
	Our Approach													
5	Single learner + manual clusters	No	54.0	73.0	62.1	55.6	74.7	63.7	57.6	77.0	65.9	30.0	72.1	42.3
		Yes	66.8	79.1	72.5	72.8	75.8	74.3	72.4	74.8	73.6	37.3	73.4	49.5
6	Single learner + induced clusters	No	53.2	73.5	61.7	54.8	73.6	62.8	55.2	75.0	63.6	30.0	73.5	42.6
		Yes	65.6	78.1	71.3	72.0	75.3	73.6	72.8	77.1	74.9	37.6	73.2	49.7
7	Two learners + manual clusters	No	61.6	84.6	71.3	69.2	80.5	74.4	71.2	82.4	76.4	35.2	72.8	47.5
		Yes	68.4	84.2	75.5	75.5	83.4	79.3	78.0	84.4	81.1	39.8	73.4	51.6
8	Two learners + induced clusters	No	62.8	81.8	71.0	68.8	82.3	74.9	71.1	83.6	76.8	35.0	73.8	47.5
		Yes	71.2	84.0	77.1	74.6	83.8	78.9	76.4	86.8	81.3	39.5	71.9	51.0

Notes: R, P, and F are denoted as Recall, Precision, and F-score respectively

Overall, these results show that (1) our 2-learner, pseudo-instances-enhanced, ontology-based approach is effective, and (2) feature selection consistently improves performance.

To gain insights into which features and which clustering algorithms output are being selected, the best-performing system (row 8 in Table 3) has selected (as determined on the dev set) the feature subsets of features 1 (same words), 3 (verb pairs), 4 (verb group pairs), and 5 (noun pairs), as well as the Interactive (with 20 clusters) clustering output.

6.2.3. FURTHER INVESTIGATION

This section discusses usage of pseudo instances during model training to boost the learning performance**.**

Using Positive Pseudo-instances

The "Pseudo pos only" column of Table 3 shows the results when each of the systems is trained with additional positive pseudo-instances.

Comparing row-wise with "No pseudo" column, it shows that employing positive pseudo-instances increases performance on Pine. F-scores rise by 1.1–3.9% without FS and 1.8–3.8% with FS. The corresponding F-scores in all cases are statistically distinguishable. These results seem to suggest that the addition of positive pseudo-instances is useful for traceability link prediction.

Using Positive and Negative Pseudo-instances

The "Pseudo pos+neg" column of Table 3 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, it shows that additionally employing negative pseudo-instances almost consistently improves performance: F-scores rise by 0.8–2.0% without FS and up to 2.4% with FS, with one exception case (1 learner with manual clusters and FS, which drops by 0.7%). Nevertheless, the corresponding F-scores in two of four 1-learner cases (manual/with FS, induced/no FS) are statistically indistinguishable. But interestingly, the improvements in F-score in all four 2-learner cases are statistically significant. These results suggest that the additional negative pseudo-instances provide useful supplementary 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 1.3–14.5%.

Finally, the best results in our experiments are achieved when both positive and negative pseudo instances are used in combination with manual/induced clusters and feature selection: F-scores reach 81.1–81.3%. These results translate to significant improvements in F-score over the supervised baseline (with no pseduo instances) by 13.4–13.6%, or relative error reductions of 41.5– 42.1%.

Pseudo-instances from Residuals

Recall that Zaidan et al. [1] created pseudo-instances from the text fragments that remain after the rationales are removed. In Section 5.2.3, an argument has arisen that their method of creating positive pseudo-instances for our requirements traceability task is problematic. In this subsection, the correctness of this claim is verified empirically.

Specifically, the "Pseudo residual" column of Table 3 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, it shows that replacing our method of creating positive pseudo-instances with Zaidan et al.'s method causes the F-scores to drop significantly by 21.0–30.3% in all cases. In fact, comparing these results with the corresponding "No pseudo" results, it shows that 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.

7. CONCLUSIONS

A 2-learner, ontology-based, pseudo-instances-enhanced approach to supervised traceability prediction has been investigated. Results showed that (1) our approach is effective: in comparison to the best baseline, relative error reduces by 56.0%; (2) the pseudo instances extension is effective, which is able to mitigate the situation when human labelled links are insufficient; and (3) also interestingly, results obtained via induced clusters were as competitive as those obtained via manual clusters, which indicates potential to automate building of ontology rationales for traceability prediction and reduce human effort.

REFERENCES

[1] O .Zaidan, J. Eisner, and C. Piatko,"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, 2007, pp. 260–267.

[2] R.A. Pierce, "A requirements tracing tool," ACM SIGSOFT Software Engineering Notes, vol. 3, no. 5, pp. 53–60, 1978.

[3] J. Jackson, "A keyphrase based traceability scheme," in Tools and Techniques for Maintaining Traceability During Design, IEE Colloquium on. IET, 1991, pp. 2–1.

[4] S. K. Sundaram, J. H. Hayes, and A. Dekhtyar, "Baselines in requirements tracing," ACM SIGSOFT Software Engineering Notes, vol. 30, no. 4, pp. 1–6, 2005.

[5] M. Lormans and A. Van Deursen, "Can LSI help reconstructing requirements traceability in designandtest?" in Software Maintenanceand Reengineering, 2006. CSMR 2006.Proceedings of the 10th European Conference on. IEEE, 2006, pp. 10–pp.

[6] A. DeLucia, F .Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 16, no. 4, p. 13, 2007.

[7] De Lucia, R. Oliveto, and G. Tortora, “Assessing IR-based traceability recovery tools through controlled experiments,” Empirical Software Engineering, vol. 14, no. 1, pp. 57– 92, 2009.

[8] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou, “Utilizing supporting evidence to improve dynamic requirements traceability,” in Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on. IEEE, 2005, pp. 135–144..

[9] D. Port, A. Nikora, J. H. Hayes, and L. Huang, “Text mining support for software requirements: Traceability assurance,”in System Sciences (HICSS), 2011 44th Hawaii International Conference on. IEEE, 2011, pp. 1–11.

[10] J.N. Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson, “A feasibility study of automated natural language requirements analysis in market-driven development,” Requirements Engineering, vol. 7, no. 1, pp. 20–33, 2002.

[11] Z. Li, M. Chen, L. Huang, and V. Ng, “Recovering traceability links in requirements documents,” in Proceedings of the Nineteenth Conference on Computational Natural Language Learning, 2015, pp. 237–246.

[12] J. Guo, J. Cheng, and J. Cleland-Huang, “Semantically enhanced software traceability using deep learning techniques,” in 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), May 2017, pp. 3–14.

[13] M.-C. de Marneffe, B. MacCartney, and C. D. Manning, “Generating typed dependency parses from phrase structure parses,” in Proceedings of the 5th International Conference on Language Resources and Evaluation, 2006, pp. 449–454.

[14] C.-C. Chang and C.-J. Lin, “LIBSVM: a library for support vector machines,” ACM Transactions on Intelligent Systems and Technology (TIST), vol. 2, no. 3, p. 27, 2011.

[15] A. Blum and P. Langley, “Selection of relevant features and examples in machine learning,” Artificial Intelligence, vol. 97, no. 1–2, pp. 245–271, 1997.