Software Document Terminology Recognition

Shuhei Nojiri
Yokohama Research Laboratory, Hitachi Ltd.,
292 Yoshida-cho, Totsuka-ku,
Yokohama-shi, Kanagawa 244-0817 Japan

Christopher D. Manning
Department of Computer Science, Stanford University
353 Serra Mall,
Stanford, California 94305 USA

Abstract

Our goal in this paper is to achieve automatic extraction and classification of key phrases from software development documents, such as requirements, specifications, and so on. In software development projects, creating dictionaries is important for defining the terminologies used to enable accurate communication between customers and vendors, as well as among developers. However, each target domain, such as a medical, financial, transportational, or other field, has its own particular terminology; moreover, each customer employs its own terms and their respective meanings. Building a dictionary of a target domain requires experts' knowledge in the given domain and considerable effort. To assist in dictionary building, we are developing a software document terminology recognizer (SDTR) with the use of named entity recognition (NER) methods. A significant amount of research exists on NER; however, most of it is focused on general named entities, such as person names, or biological domain named entities, such as names of compounds. However, the problem of building effective entity recognizers in a new domain where you have very little supervised data available is very understudied. There are a lot of small domains each of them has different terminology because software is used in various domains and organizations. Also it is impractical to build taggers by traditional supervised NER methods for SDTR because the tuning cost in individual software development projects is limited. Building method of an SDTR should cover cross-domain terminologies using small size of corpus; nevertheless, an SDTR must cope with very specific terminologies for individual projects. In this paper, we propose a multi-layered SDTR system consisting of an identifier that uses general features based on the probability of phrases and spelling conventions, and an identifier that employs a temporary dictionary automatically built into the general feature identifier. Currently, our prototype achieves a greater than 0.8 F1-value on a small software development project corpus.

Introduction

Automatic information extraction from software documents, such as requirements, specifications, and so on, is desirable for understanding and (re)building software, while a cross-document terminology dictionary and index are likewise desirable for software development projects. A dictionary is helpful in clarifying important terminology to foster accurate communications between customers and vendors, as well as among developers. Moreover, an index is helpful for referencing where relevant descriptions are located. Numerous studies on named entity recognition (NER) exist (Nadeau and Sekine 2007) (Leser and Hakenberg 2005) (Ratinov and Roth 2009). However Most NER research has focused on general named entities, such as person names, or biological domain named entities, such as names of compounds, the problem of building effective entity recognizers in a new domain where you have very little supervised data available is very understudied. There are a lot of small domains each of them has different terminology because software is used in various domains and organizations. Also it is impractical to build taggers by traditional supervised NER methods for SDTR because the tuning cost in individual software development projects is limited. NER is a promising, fundamental technology that enables detection of important terminology from software documents. The extracted terminology can be used for creating a dictionary to define name words and phrases, and an index that references the location where each word or phrase is explained in the particular software. Several NER systems have been recently proposed and are currently available for use, such as the Stanford NER (Finkel, Grenager, and Manning 2005). However, most of these systems have been traind and evaluated on large amounts of general domain data; it is therefore difficult to apply them for use in a software document directly for several reasons:

1. Name words and phrases depend on target domains, organizations, and projects.
2. The number of software documents for each project is limited.
3. Supervised data is lacking.
4. Most technical terms are a combination of common nouns, not proper nouns; it is difficult to distinguish them.

Figure 1 shows examples of terms excerpted from (Hitachi, Ltd. 2013). These examples focus on terms relating to "user." Four names exist that are important to the software specifications. As shown, each name is composed of common nouns; in some cases, a name combines partial phrases, such as the user ID field and user ID. The noun "user" is only employed as a common noun, such as in the third line. In addition, although a very general noun, "user," has been

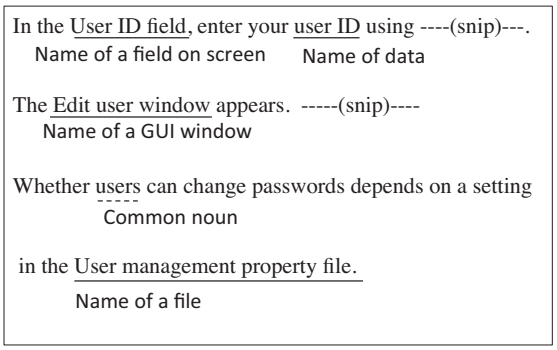

Figure 1: Examples of terms in software documents

explained, this type of challenge occurs in each domain and in each organization's lexicon. To overcome the above differences between NER for general use and NER for software documents, we are developing a software document terminology recognizer (SDTR).

Software Document Terminology Recognition

Based on NER technologies, an SDTR is a type of NER system that is specialized for software documents. An SDTR recognizes the names of software elements, such as actors, roles, dates, functions, class names, and so on, from software documents published in particular organizations. An NER, on the other hand, recognizes proper names, such as organizations, persons, places, and so on, from general publications. Owing to the above difference between SDTR and NER, it is difficult to apply existing and distributed NERs to software document directories. Table 1 shows the results of applying the Stanford NER to a software document with no changes. The tag "BN" is attached to the word of a name beginning, the tag "IN" is attached to the word of a name following BN if the name is a phrase. Unfortunately, the BN and IN recall scores are not good because the language model of the Stanford NER (Finkel, Grenager, and Manning 2005) is trained for proper names, not for names of software elements. We must therefore train a language model that is suitable for SDTR.

Tag	Precision	Recall	F1	Support
-	0.73	0.99	0.84	6539
BN	0.81	0.14	0.24	1238
IN	0.73	0.05	0.09	1502
Average	0.74	0.73	0.64	9279 (Total)

Table 1: Name tagging results using the Stanford NER

However, two contradictory problems in training an SDTR language model exist. The first problem is the difficulty of building a generalized model. This challenge is founded in the nature of software development as a cross-domain activity. Each software development project is domain- and organization-specific; moreover, the respective vocabulary and terminologies are unique. Thus, it is difficult to build a cross-domain language model. Meanwhile, the consumable costs of each software development project are limited. It is therefore challenging to remunerate the significant costs for making supervised data as well as for project-specific features in individual projects. An individual software development project requires a versatile and non-supervised SDTR; nevertheless, the challenges remain, as noted above. Thus, the SDTR should minimize the preparation of supervised data and modification of features. To solve these problems, we hypothesize the following:

- Domain- and organization-specific knowledge can be eventually implemented as a list of names.
- Because the essential types of names relating to software specifications such as actors, functions, and data can be unified, a method can be developed to enable the sharing of name candidates between particular projects.

If the first hypothesis is realized, the project-specific features can be implemented, just like a dictionary matcher, for each project. If the second hypothesis is realized, the supervised data can be gathered and used to train greater data than individual training. Based on the above hypotheses, we propose a two-phased SDTR, which is elaborated in the following sections.

Research Questions

Outlined below are the three SDTR research questions of this paper.

RQ1 Is using a project-specific name candidate list effective for the F1-value of the SDTR?

RQ2 Can it build the name list based on general features; i.e., domain-independent features?

RQ3 Is the automatically built name list helpful in improving the SDTR?

Phased Software Document Terminology Recognizer

We propose a two-phased SDTR that is (semi)supervised. Figure 2 depicts its architecture. The names included in its required training document should be tagged "BN" and "IN." After training, the SDTR receives non-tagged target documents and then outputs named tagged documents. These two phases are described below.

First phase: Creating the Name Candidate List

The objective of the first-phase tagger is to create a name candidate list for a particular software project for the second-phase tagger. High recall is an important criterion for first-phase taggers, even though it sacrifices precision. Because a name candidate list is expected to function as domain-specific knowledge of target documents, it is preferable that the list includes all names in target documents. In addition, first-phase taggers should comprise domain- or organization-independent features because first-phase taggers are expected to be trained to improve recall by using cross-domain or cross-organization data. Each software

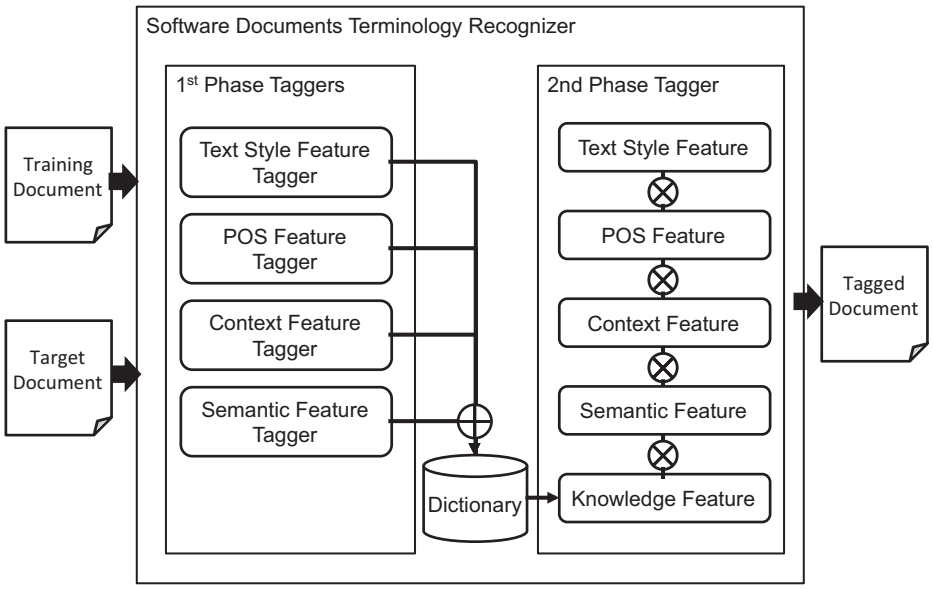

Figure 2: Architecture of the two-phased SDTR

development project includes domain- and organization-specific terminology, as mentioned earlier. Therefore, if a tagger is trained as a project-specific tagger, it constitutes over-training when reusing it for other project documents. By omitting domain- or organization-specific features, these taggers can be trained by a large amount of cross-project data to obtain a high-quality name candidate list. Some non-domain- and non-organization-specific features are employed in four feature aspects: text style, structure, context, and semantics. To build a high-recall name candidate list, four taggers are composed using each of the feature aspects. The four taggers individually tag names to input documents; the SDTR then detects the names from the tagged documents and inputs them into the name candidate list.

Text Style Features Text style features of written letters include capitalization, prefixes, suffixes, bold facing, italicization, and so on.

Structure Features Structure features focus on structural information of the word in a sentence. One of the most common structure features is the part of speech. The position of a word in a sentence, such as at the beginning or end, or its relative order, could be a structure feature.

Context Features Context features focus on the relationship of neighbor words. Features based on n-gram (Brown et al. 1992) probability have often been used in previous studies.

Semantic Features Semantic features focus on word meanings. It is difficult to directly express a words meaning; therefore, a word vector and group of synonyms may be substituted for the expressed meaning.

Second Phase: Performing Project-specific Tagging

The second-phase tagger provides conclusive tagging to the target documents using the name candidate list that was output in the first phase. The role of the second phase is to output name-tagged documents to SDTR users. This phase therefore requires both high recall and high precision. Accordingly, the criterion of the second-phase tagger is the F1-value. To obtain a high F1-value result, both project-specific and non-project-specific features are employed for the second-phase tagger. Ideally, it is desirable to calculate the project-specific features based on domain and organization knowledge analyzed by experts. However, this is labor-intensive. We do not employ manually created knowledge resources; rather, we attempt to use the name candidate list from the first-phase tagger. For our second phase tagger, we combine those five feature into one tagger.

Knowledge Features Knowledge features focus on the word or phrase is known to or used as a entity name in particular domain. Matching with the name list, such as a dictionary, could be an implementation of these features.

Experiments

To evaluate the applicability of our proposed SDTR, we built prototypes of the first-phase and second-phase taggers. We employed Python 2.7.3 and the Python-crfsuite (Peng and Korobov 2014), which is a Python binding to CRFsuite (Okazaki 2007), as an implementation of conditional random fields (Lafferty, McCallum, and Pereira 2001) for these prototypes. For the sample software document, we used a software user manual (Hitachi, Ltd. 2013) from which we excerpted Chapters 1, 2, and 3 and manually tagged names. Table 2 shows a summary of the sample data. In this

paper, we report the results with Chapter 2 used for supervising data and with Chapter 3 used for test data.

Chapter	Number of Words	Remarks
1	3586	
2	859	training data
3	9279	test data

Table 2: Size of the training and test data

The feature setups that we built for the prototypes are described below.

Text Style Features: Capitalization The text style features of our prototypes are the following:

- If the target word is capitalized.
- If the next word is capitalized.
- If the previous word is capitalized.

Structure Features: Part of Speech The part-of-speech structure features of our prototypes are the following:

- Part-of-speech tag of the target word.
- Part-of-speech tag of the next word.
- Part-of-speech tag of the previous word.

For calculating the part of speech, we employ a part-of-speech tagger of the Natural Language Toolkit (NLTK) (NLTK Project 2014).

Context Features: BM25 Scores of Uni- and Bi-grams The following are context features of our prototypes:

- Normalized and step-wised values of the uni-gram BM25 (Robertson, Zaragoza, and Taylor 2004) score for the target word.
- Normalized and step-wised values of bi-gram BM25 scores for the target word and neighbor words.

Semantic Features: Similar Words Based on Word Vector The following are semantic features of our prototypes:

- The top ten similar words of the given term.
- The top ten similar words of the neighbor words, the next two words, and the previous two words.

For calculating similarity, we employed WordVector (Mikolov et al. 2013); we used word2vec (word2vec project) as an implementation of WordVector. In addition, we used the sample document for the corpus of calculating WordVector. Accordingly, the corpus size was 13,724 words.

Knowledge Features: Matching with Name Candidate List The following are knowledge features of our prototypes:

- If an exact phrase match, including the target word, with a window size up to five, exists in the name candidate list.
- If a partial phrase match, including the target word, with a window size up to five, exists in the name candidate list.

Experiment One

To examine the utility of the project-specific name candidate list, we first composed second-phase taggers using an ideal name candidate list and applied them to our test data. The ideal name candidate list included all name words and phrases extracted from the sample document that we tagged.

Table 3 shows the results of the oracle second-phase tagger which use an ideal name candidate list. The first line presents the results of applying a tagger implemented only with knowledge features; i.e., feature matching with the name candidate list. The second line presents the results of using a tagger that combines text style, structure, context, semantic, and knowledge features. The third line provides the results of the best mixed tagger of this experiment. This tagger incorporated text style, structure, semantic, and knowledge features, but not context features.

Experiment Two

We then applied to our test data the prototype of the first-phase tagger using the above features. The purpose of this experiment was to examine the feasibility of automated name candidate list extraction based on our proposed architecture. Table 4 shows the statistical results of each first-phase tagger.

After the completion of name tagging, we created the name candidate list from each result tagged document by extracting the name words and phrases and eliminating duplications. We then obtained the first-phase name candidate list by merging the four name candidate lists. Table 5 shows a summary of extracted names candidate lists. The first four lines present the results of elemental taggers; the bottom line presents the results of the merged name candidates list.

Source	Correct	FN	FP
Text Style: Capitalize	175	281	361
Structure: Part of Speech	158	298	174
Context: 1,2-gram BM25	173	283	310
Semantics: Similar Words	172	284	271
Merged	175	281	361

Table 5: Summary of extracted name candidate list using each first-phase tagger. FN denotes false negatives; FP represents false positives.

Experiment Three

Finally, to evaluate the SDTR performance, we applied the prototypes of the second-phase tagger using the name candidate list, which was extracted by the first-phase taggers in Experiment Two. We attempted all combinations of the above features and compared the performances. Table 6 shows the statistical results of the second-phase tagger prototypes. The first line presents the results tagged by only knowledge features; the second line presents the results by the tagger that included all features implemented at this time; and the bottom line presents the best result tagger that included every feature except context features.

Features	-			BN			IN			Average		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Only Knowledge	0.94	0.97	0.95	0.87	0.64	0.74	0.89	0.98	0.93	0.92	0.93	0.92
Use all	0.95	0.92	0.96	0.85	0.70	0.77	0.93	0.96	0.95	0.93	0.93	0.93
Without Context	0.95	0.97	0.96	0.88	0.71	0.78	0.93	0.97	0.95	0.94	0.94	0.94

Table 3: Results of the oracle second-phase tagger. P denotes precision, R denotes recall, and F1 denotes the F1-value.

Features	-			BN			IN			Average		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
TextStyle: Capitalize	0.77	0.94	0.84	0.69	0.30	0.42	0.54	0.27	0.36	0.72	0.74	0.71
Structure: Part of Speech	0.91	0.91	0.90	0.70	0.60	0.65	0.74	0.85	0.79	0.85	0.85	0.85
Context: 1,2-gram BM25	0.71	0.93	0.80	0.30	0.08	0.13	0.21	0.05	0.08	0.57	0.67	0.60
Semantics: Similar Words	0.86	0.95	0.90	0.69	0.43	0.53	0.79	0.67	0.72	0.83	0.84	0.72

Table 4: Tagging results of each first-phase tagger. P denotes precision, R denotes recall, and F1 denotes the F1-value.

Evaluation

In this section, we evaluate our research questions based on the results obtained by the above experiments.

For RQ1

If a suitable list is provided, using a project-specific name candidate list is effective. According to Table 3, the best average F1-value was 0.94. Even though the tagger used only knowledge features, the average F1-value was 0.92. This result implies that the features using project-specific knowledge, such as our name candidate list, are effective for SDTR.

However, when focusing on the BN results, it is evident that the recall scores were not very high; i.e., up to approximately 0.71. This means that our current SDTR missed 30% of the beginning of names. This adversely affected the performance of automated name extraction; therefore, we must develop supplemental features that can improve BN recall.

For RQ2

It seems feasible to build the name candidate list using general features, that is, the first-phase taggers in our SDTR. However, the performances of the first-phase prototypes were not sufficient. According to Table 5, the final merged name candidate list had 145 correct names; however, it missed 284 names. That is, the recall of the extracted name candidate list was not much more than 40There are three types of extraction errors: missing, over-extracted, and under-extracted. Missing errors occur when the tagger cannot tag any words of a name; over-extraction occurs when the tagger tags one or more excess words of the head or tail of a name; and under-extraction occurs when the tagger tags a segment shorter than the radical length of a name. By analyzing the extracted name candidate list, it was determined that the list included several over- and under-extractions. These extraction errors caused a decrease in recall. Hence, we require further development of features that can cope with over- and under-extractions. In addition, a re-design and re-examination of the first-phase tagging architecture are required. In designing the first-phase taggers for this paper, we employed a structure that uses four plain taggers and merges each extracted name into the name candidates list, as shown in Figure 2. We presumed that the performances of each tagger would not be very high; however, we expected that the combination of the results from different viewpoints could improve the recall. Nevertheless, belying our expectation, one feature dominated. In this experiment, the name candidate list by text style features contained all name candidates of the three other features. Consequently, the top and bottom records of Table 5 present the same values.

For RQ3

Even though the automatically extracted name candidates list did not have high recall, the results of our second-phase tagger seem meaningful. According to Table 6, the best average F1-value was 0.87. For the tagger using only knowledge features, the average F1-value was 0.82. Given that the extracted name candidate list included only 40% of correct names, these results are surprisingly high. We estimate that is because there were a number of over- and under-extracted name candidates in the name candidate list; furthermore, the knowledge features matched with those nearly names like a correct names.

Discussion

In this section, we explain the findings from our experiments.

Part-of-Speech Features

It is noteworthy that the part-of-speech features performed well with respect to their plain and common setups. As shown in Table 4, the results of "Structure: Part of Speech" do not appear considerably weak against the final results presented in Table 6. Understandably, the results of the elaborated features were better than those for the part-of-speech features. However, the setup of the features was very simple and current part-of-speech taggers are maturing. In addition, (Chuang, Manning, and Heer 2012) also reported usefulness of part-of-speech patterns for extracting terminolog-

	-			BN			IN			Average		
Features	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Only Knowledge	0.90	0.88	0.89	0.64	0.56	0.60	0.66	0.80	0.73	0.83	0.82	0.82
Use All	0.92	0.90	0.91	0.69	0.66	0.67	0.76	0.85	0.80	0.86	0.86	0.86
Without Context	0.92	0.91	0.92	0.70	0.68	0.69	0.77	0.85	0.80	0.87	0.87	0.87

Table 6: Results of the second-phase tagger using the name candidate list extracted by the first-phase taggers. P denotes precision, R represents recall, and F1 denotes F1-value.

ical phrases. Thus, it seems it would be sufficiently helpful to use in further trials the software documents of some development projects.

Word Vector Feature

The performances of the semantics features, i.e. similar words calculated by WordVector, were also noteworthy. In most related works on WordVector, the corpus sizes were millions or more; however, we provided a mere 13,000 words in this paper. Although this was an insufficient amount of corpus, the semantics features worked well, as demonstrated by the result of "Structure: Part of Speech" in Table 4. For future work, we will prepare a large software document corpus and research the performances of WordVector features.

Context Features

Context features, with a uni-gram and bi-gram BM25 score, unfortunately did not seem to work well in our experiment. As evidenced in Tables 3 and 6, the context features degraded the performances in both cases. We suspect that the n-gram window size was not sufficiently large and that the training data amount was inadequate. The cause will be investigated in future work.

Conclusion

In this paper, we presented a two-phased SDTR in which the first phase extracts the name candidate list using general features; the second phase provides tagging using the extracted name candidate list as project-specific knowledge. Based on experimental results, we confirmed the availability of the features using the name candidate list and achieved an average 0.87 F1-value of name tagging using the extracted name candidate list. Nevertheless, the performance of the name candidate list extraction was insufficient and requires further study. Moreover, the experiments in this phase were only for proof of concept; thus, the test data size was small. In future work, we will conduct experiments based on a greater amount of test data.

References

Brown, P. F.; Desouza, P. V.; Mercer, R. L.; Pietra, V. J. D.; and Lai, J. C. 1992. Class-based n-gram models of natural language. *Computational linguistics* 18(4):467–479.

Chuang, J.; Manning, C. D.; and Heer, J. 2012. "without the clutter of unimportant words": Descriptive keyphrases for text visualization. *ACM Trans. on Computer-Human Interaction* 19:1–29.

Finkel, J. R.; Grenager, T.; and Manning, C. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, 363–370. Stroudsburg, PA, USA: Association for Computational Linguistics.

Hitachi, Ltd. 2013. *Job Management Partner 1/Integrated Management - Service Support Operator's Guide*. Hitachi, Ltd., 3021-3-365(e) edition.

Lafferty, J. D.; McCallum, A.; and Pereira, F. C. N. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, 282–289. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Leser, U., and Hakenberg, J. 2005. What makes a gene name? named entity recognition in the biomedical literature. *Briefings in Bioinformatics* 6(4):357–369.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nadeau, D., and Sekine, S. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30(1):3–26.

NLTK Project. 2014. Natural language toolkit. http://www.nltk.org/index.html.

Okazaki, N. 2007. Crfsuite: a fast implementation of conditional random fields (crfs). http://www.chokkan.org/software/crfsuite/.

Peng, T., and Korobov, M. 2014. python-crfsuite. http://python-crfsuite.readthedocs.org/.

Ratinov, L., and Roth, D. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, 147–155. Association for Computational Linguistics.

Robertson, S.; Zaragoza, H.; and Taylor, M. 2004. Simple bm25 extension to multiple weighted fields. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, 42–49. New York, NY, USA: ACM.

word2vec project. word2vec. https://code.google.com/p/word2vec/.