International Journal of Advanced Science and Technology
Vol.112 (2018), pp.123-136
http://dx.doi.org/10.14257/ijast.2018.112.12

Improving Classifiers for Semantic Annotation of Software Requirements with Elaborate Syntactic Structure

Yeongsu Kim[1,2], Seungwoo Lee[1,2], Markus Dollmann[3,4] and Michaela Geierhos[3,4]

[1]University of Science and Technology (UST), Korea
[2]Korea Institute of Science and Technology Information (KISTI)
[3]Heinz Nixdorf Institute
[4]University of Paderborn
kys5670i@ust.ac.kr, swlee@kisti.re.kr,
{dollmann|geierhos}@hni.upd.de

Abstract

A user generally writes software requirements in ambiguous and incomplete form by using natural language; therefore, a software developer may have difficulty in clearly understanding what the meanings are. To solve this problem with automation, we propose a classifier for semantic annotation with manually pre-defined semantic categories. To improve our classifier, we carefully designed syntactic features extracted by constituency and dependency parsers. Even with a small dataset and a large number of classes, our proposed classifier records an accuracy of 0.75, which outperforms the previous model, REaCT.

Keywords: *Software Engineering, Natural Language Processing, Semantic Annotation, Machine Learning, Feature Engineering, Syntactic Structure*

1. Introduction

By virtue of hardware and software developments, computer science (CS) has become a popular research discipline over the past several decades. This phenomenon is confirmed in academic as well as industry research. The number of papers related to CS has been increasing since 1990 based on the statistics of the Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE) [1], and Information Technology (IT) companies are leading the industrial market. Owing to the fact that the CS domain is growing explosively, the complexity of software programs is steadily increasing, and toolkits such as program languages are also evolving. This fact presents some challenges, one of which is the facility of communication between developers and users. Developers use explicit language in order for computers to understand, whereas users typically speak in abstract forms assuming that they understand each other's context. Therefore, users generally write software requirements using natural language, which has certain limitations in its application to computers such as ambiguity, over-flexibility, amalgamation, confusion and lack of modularization [2]. These limitations have led to the evolution of natural language processing (NLP), which is a field concerned with the interactions between computers and humans. If we think of software developers as computers, the communication problem can be solved using NLP.

In the field of software engineering, Dollmann (2016) mentioned that NLP has been defined as semantic annotation (SA) in which each word in a software requirement text can be annotated with semantic categories. With semantic categories pre-defined by researchers, we are able to deal with communication barriers such as ambiguous and incomplete expressions. In practice, as long as software requirements are semantically

Received (November 8, 2017), Review Result (January 25, 2018), Accepted (January 31, 2018)

annotated, developers can quickly figure out requirements, or give feedback to users when their requirements are incomplete, and again after the users restate their requirements in a clear form. Furthermore, developers are able to understand users' requirements even though they are ambiguous. Therefore, the goal of this study is to bridge the communication gap between developers and users by using NLP techniques such as machine learning methods and semantic categories. Furthermore, our model that sensitively designed on this task outperforms the previous model, REaCT [3].

2. Related Work

Two representative approaches – the rule-based approach and the statistical machine-learning (ML) approach – have been adopted for NLP. In general, the rule-based approach is a traditional method incorporating domain knowledge, and is used when a large dataset could not be involved. The ML approach is a way to leverage the power of statistics, and it is used when a large dataset is involved. Both have their respective pros and cons [4]. The rule-based approach is easy to understand and maintain because its rules are designed on human knowledge; however it is not robust, and it requires considerable human effort. In contrast, the ML approach is more robust because it is adaptable and trainable and it does not need much human effort; however, it requires a large amount of data and certain ML technologies. A distinct difference between the two approaches is that the rule-based approach displays explicit action behavior, whereas the ML approach relies on implicit action behavior based on features whose weights are updated by the model and the data. For example, Palmer (1990) considered taxonomies as explicit thresholds for decision-makings, whereas Liang (1992) used them as additional features with flexible weights for soft decisions. Because the ML approach includes some rules as features to be learned from the dataset, the ML approach is more general than the rule-based approach.

In the NLP domain, there are many different types of tasks including document classification, named entity recognition, semantic role labeling, and machine translation. The NLP task that is most similar to our problem, SA, is semantic role labeling in terms of identifying semantic roles and their arguments in software requirements [3]. Because such roles and arguments are too complex in form to be described with explicit rules, meaning that there are many exceptions when using a rule list, the current approaches to semantic role labeling are mostly based on supervised machine learning models with various types of feature sets [7]. Given this knowledge, we adopted an ML approach for solving our problems as the previous research did. In order to represent complex information, we need to understand other NLP tasks such as parsing, chunking, part of speech tagging, and named entity recognition, because for our classifier, we concatenate features identified by other NLP classifiers [8]. Hence, it is essential to understand not only semantic role labeling but also other NLP tasks for solving our SA problem.

3. Semantic Annotation

Given the fact that words should be annotated with categories, we consider our SA problem as a classification problem with semantic categories; therefore, we use supervised ML models. Then, what are the semantic categories? They are represented in Figure 1e, and proposed by Dollmann (2016), whose authors manually defined them with knowledge of software engineering and frames obtained after examining the entire dataset. The authors believed that even though text forms of software requirements can be expressed in different ways, they could be grouped into semantic and relevant regions in the sentence, such as description and functionality. In other words, a software requirement can be semantically decomposed into several categories according to the relevant content. For example, *component* usually refers to a product or system and *action* represents what a component affects. Because some software requirements contain additional or refinement descriptions and subordinate clauses, there exist certain types of modifier categories, such as *refinement of component* and *argument of action*. A single example of a software requirement is presented in Figure 1a.

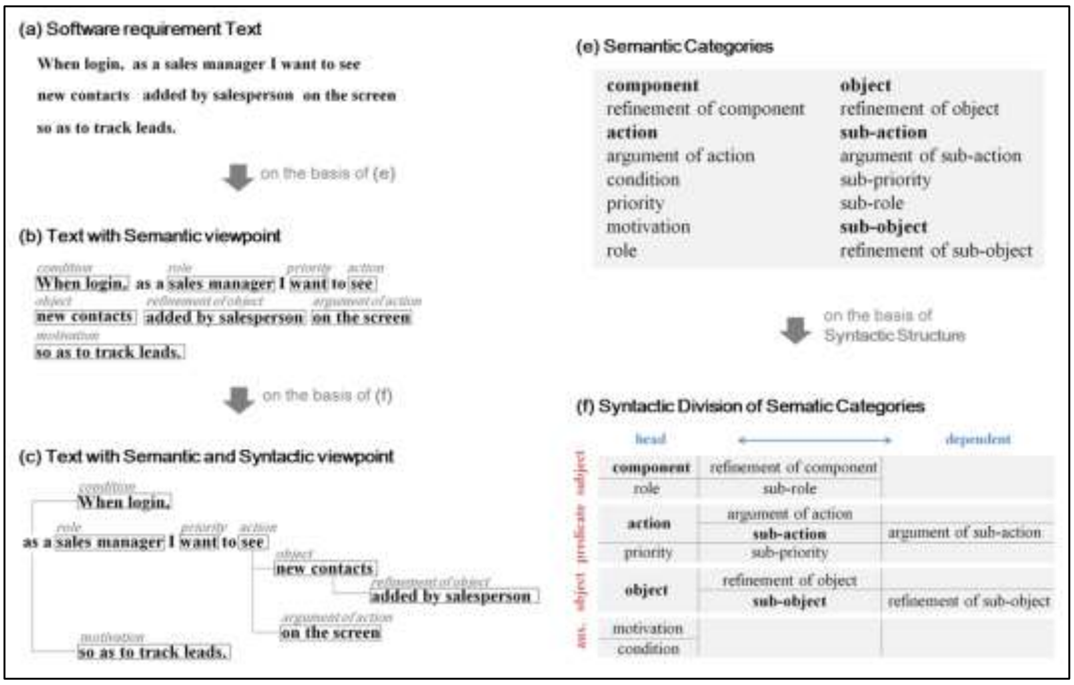

Figure 1. Semantic Category with Syntactic Structure

Given the semantic categories, the requirement can be semantically annotated as in Figure 1b. We can easily understand what the meaning of the software requirement is after it is annotated. One linguistic theory suggests that a language is a set of levels of representations such as phonemics, morphology and phrase structure [9], which is a pervasive approach in NLP today. Given this knowledge, semantic categories can be syntactically divided into groups in two ways: based on grammar or hierarchy (Figure 1f). With the grammar aspect, the categories are grouped into subject, predicate and object, which are the principal components of the sentence. In addition, categories are represented by a hierarchical tree structure, where each category is arranged on the basis of dependence. For example, *sub-object* is considered as a head because *refinement of sub-object* is dependent on *sub-object*, and at the same time, *sub-object* can be a dependent because it depends on *object*. Head and dependent are usually composed of noun or verb and phrase or clause, respectively. Finally, software requirement text can be both syntactically and semantically framed, as in Figure 1c. Note that such a frame is closely related to results from the sentence extracted by a parser for NLP, but is not exactly same. The important thing is that if our classifier is able to understand the frame, its performance will increase because of its relevance. Thus, we carefully design elaborate syntactic features based on the frame.

4. Our Approach

Our strategy both to annotate requirements and to improve the previous model involves three tackle parts: data, feature and model part. They are not just the essential parts for building machine learning pipeline but also the source of prediction errors. Also, they are less correlated with each other. When we build a supervised learning classifier, assuming that we have mean squared error criterion and irreducible error ε is omitted, the total prediction error is defined as

$$\underbrace{\sum_{i=0}^{m}\frac{1}{m}\left[\left(\boldsymbol{y}_i-\hat{f}(\boldsymbol{x}_i)\right)^2\right]}_{total\ error}=\underbrace{\sum_{i=0}^{m}\frac{1}{m}\left[\hat{f}(\boldsymbol{x}_i)-f(\boldsymbol{x}_i)\right]}_{bias\ error}+\underbrace{\sum_{i=0}^{m}\frac{1}{m}\left[\left[\hat{f}(\boldsymbol{x}_i)^2\right]-\left(\left[\hat{f}(\boldsymbol{x}_i)\right]\right)^2\right]}_{variance\ error}$$

where $\boldsymbol{x}_i \in \mathcal{R}^d$ is a d dimensional feature vector for a certain software requirement (Section 4.2), $\boldsymbol{y}_i \in \mathcal{R}^{17}$ is a true label with one-hot encoding composed of 16 semantic classes and a noise class, $\hat{f}(\boldsymbol{x})$ is function described by our learned classifier, and $f(\boldsymbol{x})$ is a true function. The total error is decomposed into bias error and variance error. Bias error, as the distance from true function, occurs when the model $\hat{f}(\boldsymbol{x})$ is not sufficient to explain the distribution of the dataset and the feature vector $\boldsymbol{x}$ is not representative in distinguishing all the classes. We decrease the bias error by selecting the appropriate machine learning model $\hat{f}(\boldsymbol{x})$ among various types of models to specific dataset, by tuning hyperparameters of the model $\hat{f}(\boldsymbol{x})$, and by designing the representative feature vector $\boldsymbol{x}$. Variance error arises when the dataset is inherently noise and fluctuant. We deal with the high-variance problem by cleaning a noisy dataset and by performing normalization to avoid sparsity and collect more representative data. High-variance can be solved by model selection and feature engineering as was the high-bias problem, but in addition, the data characteristics cause high-variance intrinsically. Note that there is a trade-off problem between high bias (*i.e.* underfitting) and high variance (*i.e.* overfitting) and therefore, we need to find a consensus. Given this common knowledge in machine learning research, we build a machine learning pipeline that includes data, feature and model part (Figure 2).

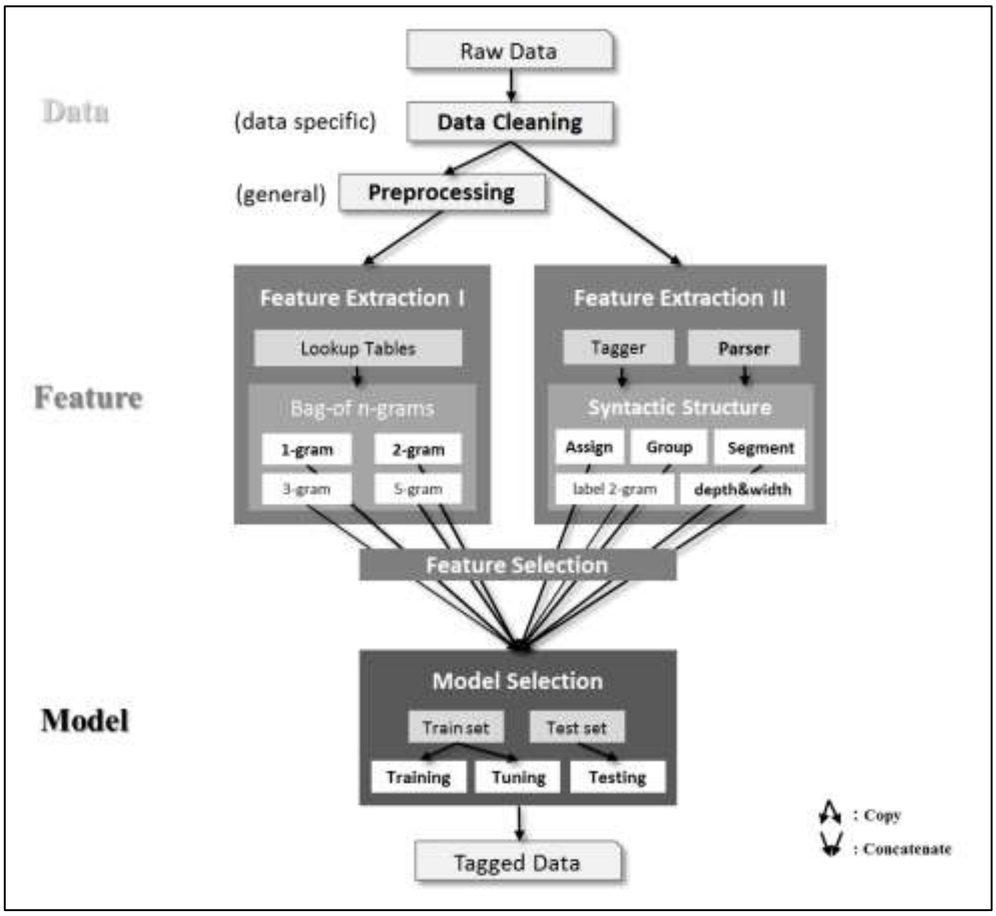

Figure 2. Machine Learning Pipeline

4.1. Data Cleaning and Preprocessing

Because a text dataset is discrete and sparse, it is important that words that have the same stem are grouped into a single representative word, and that low entropy words such as unique words and stop-words, are removed. Because our text data, as software requirements, have a specific type and form, we performed specific preprocessing during data cleaning. For example, redundant but frequent words and phrases such as *Hello, Also, Hi, In addition, I think, In general, i.e., e.g.* were removed. We changed the word, *if* into *that* when we encountered the pattern, *it would be X if*. We converted imperative sentences into declarative sentences by adding an impersonal pronoun. Some prepositions were removed when they were with intransitive verbs. Some special characters such as the hyphen and the slash were removed. In the preprocessing step, as general text lowering and stemming were executed, but we did not remove stop words because it helped bag-of-n-gram features (4.2 Feature Engineering). We chose stemming instead of lemmatization for higher performance because lemmatization feeds the parser poor text forms. We did not perform any text preprocessing for syntactic feature extraction because this may cause poor parser performance. This is the reason why we have two types of preprocessing: data cleaning and preprocessing. We use the Natural Language Toolkit (NLTK)[2] Python library to implement these steps. Data cleaning and preprocessing are important not just for mitigating the sparsity problem, but also for improving the performance of constituency and dependency parsers, whose outputs are used for syntactic features.

4.2. Feature Engineering

Our challenge is to train a classifier, and to be able to distinguish a large number of classes within a small dataset (5.1 Dataset). Given these conditions, feature engineering is considered as the most important step in the pipeline. In general, there are two representative insights when designing or selecting the feature set: data appearance and domain knowledge, which are included in the check list for solving feature selection problems as described in [10]. First, in terms of data appearance some frequent patterns exist in the dataset, such as *i want to be, it would be nice, job seeker can, the product shall, user can, a user,* and *the product*. A bag-of-n-grams provides good representations for recognizing these patterns. We build a bag of n-grams with several levels, whose top 10 words are listed in Table 1. Because all of our features are designed for each word, 2-gram, as even number, can represent two different features such as left-2-gram and right-2gram. As bag-of-n-gram features are described by lookup tables, this gives us high-dimensional and sparse feature vectors. These types of features are useful because most expressions used in software requirements are not arbitrary but specific forms to describe the purpose of the requirements.

Table 1. Top10 Word List of n-grams

1-gram	2-gram	3-gram	5-gram
the	*, i*	*, i want*	*, i want to be*
.	*as a*	*i want to*	*it would be nice to*
to	*i want*	*be abl to*	*as a user , i*
a	*want to*	*it would be*	*user, i want to*
,	*would be*	*the product shall*	*, i want to see*
be	*the system*	*the system shall*	*a user , i want*
i	*to the*	*would be nice*	*staff member shall be abl*
of	*it would*	*a user can*	*nurs staff member shall be*
shall	*a user*	*to be abl*	*member shall be abl to*
as	*abl to*	*so that i*	*it would be nice if*

[2] https://nltk.org/

Second, with domain knowledge representing the syntactic frame as we discussed in Section 3, we design elaborate syntactic features by using language resources such as the part-of-speech tagger and the constituency and dependency parser. We use the spaCy[3] Python library and Stanford Parser[4] for implementing them. With the two parsers, we represent a sentence into syntactic features in two ways. Table 2 shows syntactic features designed by both parsers. The constituency parser describes a sentence as a tree structure based on phrase-structure rules. The depth and width of the tree structure at each word can be syntactic positions. Depth is calculated via the number of parents and width through the number of siblings on the same level. With this hierarchical property, we design tree-label bigram feature, which is able to distinguish parent ambiguities. The dependency parser represents a sentence not as a tree structure, but by simple relational or dependent sets such as *nsubj*, *nmod*, and *det*. For each word, syntactic characteristics are extracted such as the part-of-speech tag, and the dependent type. By using the dependent type, words are grouped into phrases and clauses. One benefit of dependency over constituency is that dependency is used to not only group words but also recognize words, phrases and clauses in greater detail by using detailed dependency relations such as *dobj*, *pobj*, *advcl*, and *ccomp*. This means that we are able to know whether a noun is an object with a preposition or a direct object and whether a clause is a relative clause or an adverbial clause. Furthermore, a sentence is segmented based on the degree of dependencies. All features described in Table 2 are a representation vector for each word. We expect that even a non-sequential classifier can understand syntactic structure if we use the combination of feature set.

Table 2. Syntactic Features Identified by Parsers

Parser	Feature Set	Feature Description
Constituency Parser	Tree Measurement	The number of parents (depth), the number of siblings (width)
	Tree-label Bigram	With tree-label sequence from leaf to root, we build tree-label bigram in order to resolve the ambiguity of the parent node (e.g., SBAR-NP, SBAR-VP, PP-PP, PP-NP, *etc.*)
Dependency Parser	Token Characteristic	A token is assigned to its characteristics, such as part-of-speech tag, dependent type, the number of dependencies, preposition type, distance to root, its head part-of-speech tag and dependent, whether it is right/left-side of root/sub-root, *etc.*
	Group Information	Tokens are grouped into noun, prepositional phrases and relative, or adverbial clauses, and their type (e.g., for, in, if, since, that, what, when, *etc.*) and their position (e.g., is the first/last, is the right/left-side of root, *etc.*). Further, a sentence is segmented into three parts based on the root and sub-root word determined by the number of dependencies

As the whole feature vector is high-dimensional and sparse, it is worthwhile using dimension reduction techniques such as principal component analysis (PCA) and univariate selection with statistical tests. However, dimension reduction techniques were ineffective on our classifier, meaning that all information in the feature vector is important. In comparing the models with various feature sets in the 3-gram and 5-gram in

[3] https://spacy.io/
[4] https://nlp.stanford.edu/software/

bag-of-n-gram features, with the tree-label bigrams in the syntactic feature sets, low performance of the classifier was observed. There are several reasons for this: because we do not have a large dataset, 3-grams and 5-grams may cause overfitting. Tree-label bigram features are similar to group information feature sets; they may be correlated.

These syntactic features are similar to the feature set for semantic role labeling tasks [12], [13], [14] because the goal of semantic role labeling is analogous to ours in the sense of finding events and their arguments. However, our syntactic features are more elaborate in that there are many types of group information and position information for each token, which is relevant to our problem because semantic categories are closely related to such information. In addition, we design the bag-of-n-gram features. Data preprocessing for our specific problem also helps in designing robust syntactic features because it gives a cleaner input form to parsers.

Table 3. Machine Learning Model List

Linearity	Category	Model	Hyperparameter
Linear	-	Logistic Regression (LR)	C=0.4
	-	Passive Aggressive (PA)	C=2000
Nonlinear	-	Naïve Bayes (NB)	-
	-	K-Nearest Neighbor (KNN)	#neighbor=7
	-	Decision Tree (DT)	-
	-	Support Vector Machine (SVM) with kernel	C=900
	Ensemble	Extra Tree (ET)	#tree=800
		Random Forest (RF)	#tree=1500
		Voting Classifier (VC) with LR, ET, SVM	-
	Neural Network	Feedforward Neural Network (FNN)	-
		Convolutional Neural Network (CNN)	-
		Recurrent Neural Network (RNN)	-

4.3. Model Selection

In machine learning research, various classification models exist that have their own way to find the best decision boundary. Because we do not know which one is the best for our specific problem, it is worthwhile to have a candidate model set and find the best one. In Table 3, we have 12 candidate models ranging from linear to nonlinear models including ensembles and neural networks. The best model is selected based on test error comparison. It is important in model selection to optimize the hyperparameters of each model to be adapted to a specific dataset. Among the various types of hyperparameters for each model, we chose representative hyperparameters such as regularization terms from LR, PA, and SVM, and the number of trees from DT, ET, and RF, and the number of neighbors from KNN. Grid search is used for hyperparameter tuning. We found that linear models were insensitive to the change in hyperparameters, whereas nonlinear models were sensitive. This means that it is more important to optimize hyperparameters for nonlinear models. The only sequential model among our candidate model set was the RNN. As [3] mentioned, sequential models perform worse than non-sequential models because of their small datasets. Because most of our models are non-sequential models, we use the bag-of-n-gram features with a window size that includes sequential information. It is interesting that syntactic features have inherent sequential information in their own structures. For using the candidate model set and tuning the models, we use the Scikit-learn[5], Tensorflow[6] and Keras[7] Python libraries.

[5] https://scikit-learn.org/
[6] https://tensorflow.org/
[7] https://keras.io/

5. Experiment

5.1. Dataset

The labeled software requirement dataset from [3] was used for experiments. It has approximate 704 sentences and 12753 tokens, which were divided into 16 semantic categories and one none-labeled category, as shown in Table 4. The main problem with our research is that we make a classifier to distinguish quite a large number (17) of classes within a small dataset. Furthermore, the number of instances or tokens in each class are different from each other, meaning that we have an unbalanced classification problem even with a small dataset. In terms of sentences, there are certain categories with few instances, such as *refinement of component* and *sub refinement of object*. Therefore, the quality of labeled data is very important because a wrongly labeled data point can have a significant impact on the model when its weights are updated. From [3], two annotators achieved an agreement rate of 80%. This is a relatively low score for a small dataset with a skewed class distribution. One tackle point we chose was to make consistent rules and to re-annotate the labeled dataset. Some consistent rules are as follows: words in *action* and *sub-action* categories should only be verbs or verb-phrases. Words in *object* and *sub-object* should not include prepositional phrases. After re-annotating, we expected the labeled dataset to have more consistency, which reduces the fluctuation of dataset.

In general, models for a sequential classification problem can benefit from the fixed boundaries of a sequence by virtue of BIO encoding, where we split words in each class into B (Begin), I (Inside) and O (Outside). This can mitigate the complexity of patterns although the number of classes is almost doubled. Owing to the small size of the dataset, we do not use BIO encoding because we believe that some benefits may be obtained by reducing complex patterns or noise via data cleaning and preprocessing and re-annotation with consistent rules. Of the total dataset, 80% is used for training dataset, and 20% is for testing dataset.

Table 4. Number of Instances and Sentences per Category

Semantic Category	# of instances	# of sentences
component	439	308
refinement of component	143	32
action	844	687
refinement of action	1489	269
condition	836	129
priority	1162	664
motivation	657	81
role	603	300
object	1302	653
refinement of object	1409	202
sub action	134	117
sub argument of action	282	45
sub priority	50	49
sub role	52	51
sub object	194	101
sub refinement of object	277	30
none	2880	703
Total	12753	-

5.2. Measurement

Classifiers are evaluated by precision, which is the proportion of predictions that are correct, and recall, which is the proportion of correct instances, and F_1 score, which is a weighted harmonic mean of both precision and recall. Given an error matrix or a confusion matrix, where each row represents predicted classes while each column represents true classes, precision, recall and F_1 score are calculated using Equations (1), (2), and (7), respectively. They are used for an evaluation of a single class. As all classification problems, including binary classification, consist of multi-classes, an evaluation represented as a single numerical value is obtained by averaging multiple precision or recall scores for multi-classes. There are two ways for averaging: micro-averaging in Equations (3) and (4), and macro-averaging in Equations (5) and (6). The difference between them is whether the average is calculated over the number of instances or classes. Concretely, the sum of all classes' true-positives is averaged over the total number of instances in micro-averaging, whereas the sum of all classes' precisions or recalls is averaged over the total number of classes. Micro-averaged precision, recall and F_1 score are all the same because $\sum_c^C fp_c$ and $\sum_c^C fn_c$ are always the same. Micro-averaging is the same as an accuracy measurement. One disadvantage of micro-averaging or accuracy is that the performance of a class that has a small number of instances can be dominated by the performance of a class that has a large number of instances because of the sum form. Macro-averaging can deal with this problem by assigning equal weight to the performance of each class. This means that a class that has a large number of instances is advantageous when using micro-averaging, whereas a class that has a small number of instances is favorable when using macro-averaging [15]. We use both averaging measurement for our experiments because the distribution of class frequency is highly skewed. If we use only micro-averaging for our problem, we would overlook the performance of classes that has small number of instances, such as *refinement of component*, *sub priority* and *sub refinement of object*.

$$precision = \frac{true\ positives\ (tp)}{true\ positives\ (tp) + false\ positives (fp)} \tag{1}$$

$$recall = \frac{true\ positives\ (tp)}{true\ positives\ (tp) + false\ negatives\ (fn)} \tag{2}$$

$$precision_{micro} = \frac{\sum_c^C tp_c}{\sum_c^C tp_c + \sum_c^C fp_c} \tag{3}$$

$$recall_{micro} = \frac{\sum_c^C tp_c}{\sum_c^C tp_c + \sum_c^C fn_c} \tag{4}$$

$$precision_{macro} = \frac{1}{C}\sum_{c=1}^{C} precision_c \tag{5}$$

$$recall_{micro} = \frac{1}{C}\sum_{c=1}^{C} recall_c \tag{6}$$

$$F_1\ score = 2\frac{(precision)(recall)}{(precision) + (recall)} \tag{7}$$

5.3. Results and Discussion

Experiments were conducted in four scenarios including (1) test errors of all candidate models, (2) test errors of two features, (3) the trend of test errors according to data size, and (4) the comparative study between our best model and the previous model, REaCT.

The first experiment was conducted on all candidate models, aimed at choosing the best model for our specific problem and examining the characteristics of each model based on the results in Table 5. The micro-averaged F_1 score (Micro-F_1), and the macro-averaged F_1 score (Macro-F_1) are significantly different because of the skewed class distribution. In terms of statistical power, a class that has a small number of instances (small class) is harder to train than a class that has a large number of instances (large class). As Macro-F_1 accounts for such difficulty, it usually rates lower performance than Micro-F_1. Furthermore, Macro-F_1 has more complex patterns in terms of syntactic structure if a class has a smaller number of instances. SVMs showed the best performance except for those within the VC model, whereas the NB model showed the worst performance. The SVM model is well known to be robust even with a small, complex dataset because of its high-dimensional feature space by using the kernel trick. The NB model usually fails to make a reliable estimation of the probability of each class when the dataset is small. It is interesting to note that the LR obtained performance comparable to the best SVM even though it is a linear model. Given elaborately designed feature vector with syntactic structures, even the linear model was able to understand the properties of our complex dataset. However, it has been found that in the entire candidate set, the more complex models tend to have lower test errors than the less complex models. Among neural networks, RNN, as a sequential learner, outperforms the other neural networks. These results suggest that the possibility that a complex neural network can be trained well even with a small dataset.

Table 5. Test Errors of Candidate Models

Model	Micro-F_1	Macro-F_1
Logistic Regression (LR)	0.7731	0.6835
Passive Aggressive (PA)	0.7379	0.6347
Naïve Bayes (NB)	0.5369	0.4354
K-Nearest Neighbor (KNN)	0.7131	0.6091
Decision Tree (DT)	0.6918	0.5951
Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel	**0.7808**	0.6866
Extra Tree (ET)	0.7530	0.6508
Random Forest (RF)	0.7487	0.6505
Voting Classifier (VC) with LR, ET, SVM	**0.7808**	**0.6881**
Feedforward Neural Network (FNN)	0.7460	0.6465
Convolutional Neural Network (CNN)	0.7344	0.6251
Recurrent Neural Network (RNN)	0.7549	0.6646

The second experiment is the comparison of two types of feature sets: bag-of-n-grams, and syntactic features based on parsers in Figure 3(left). We use the best model, SVM, among the candidate set for this evaluations. Owing to heterogeneous insights, those two feature sets have different properties. The bag-of-n-grams does not require any domain

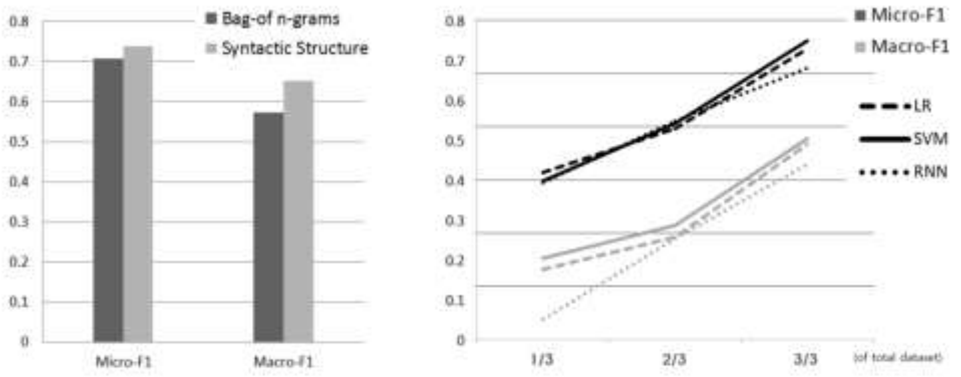

Figure 3. Test Errors of Two Feature Set and of Three Data Size

knowledge, and uses only data statistics which means it is scalable, whereas they consist of very sparse and high dimensional vector forms because of the lookup tables. Syntactic features require extensive domain knowledge which means syntactic tools are needed, whereas they are represented as dense and low dimensional vector forms. The model trained with syntactic features is significantly better than the bag-of-n-grams in both Micro-F_1 and Macro-F_1. This means that syntactic features are more important in our problem. We found that the micro-gap between two feature sets is greater than the macro-gap. We expect that because the bag-of-n-grams aggressively uses data statistics, they obtain some benefit from large classes. Instead, syntactic features are especially effective for small classes, such as *refinement of component* and *sub refinement of object*. A small class usually requires deep understanding of syntactic structures. Note that the best model is trained with both types of features. This means that two feature sets are less correlated with each other.

Our main problem is data deficiency. We expect that the performance of the models would increase with larger dataset. To verify this, the third experiment (shown in Figure 3(right)) was performed for observing performance trends as the data size increases from one-third, to the full dataset. For this evaluation, three models representing different model capacities, such as LR, SVM and RNN, were used. We found that both the Micro-F_1 and the Macro-F_1 of all models increased as the datasets became larger. Therefore, we expect to obtain better results with larger datasets.

The last experiment was for the comparison with the previous model, REaCT [3]. To ensure robustness we performed the experiment 10 times on each case with different random seeds from 1 to 10 with random shuffling dataset. Furthermore, to be fairness we used the same dataset with the same hyperparameter tuning methods. Unlike before, we show both averaged performances and each category performance in more detail (Table 6). Our proposed model significantly outperformed the previous model, REaCT. The performances of most categories and both Micro-F_1 and Macro-F_1 are improved by our proposed model. The Macro-F_1 gap between the two models (0.03) is larger than that of Micro-F_1 (0.08). This means that our proposed model works well not just with large classes but also small classes, which require more complex syntactic structures in particular. Given that it has highest standard deviation (the numerical values in parentheses) and the lowest F_1 score in the previous model, *sub refinement of object* is considered as the most difficult of the small classes to classify. When evaluating our proposed model, it was found that *sub refinement of object* was the most improved F_1 score from 0.11 to 0.49. This improvement is due to the elaborate syntactic structures of our proposed model. When we add segment id features in the group information feature set, large improvements occur. Moreover, we found that small classes such as *refinement of component*, and *sub role* were largely improved by our proposed model.

Table 6. Test Errors of REaCT and Proposed Model

Semantic Category	REaCT	Proposed
component	0.77 (0.04)	0.83 (0.04)
refinement of component	0.21 (0.13)	0.36 (0.16)
action	0.85 (0.02)	0.85 (0.01)
refinement of action	0.53 (0.04)	0.58 (0.05)
condition	0.45 (0.12)	0.73 (0.04)
priority	0.92 (0.02)	0.94 (0.01)
motivation	0.70 (0.09)	0.75 (0.03)
role	0.91 (0.02)	0.96 (0.00)
object	0.76 (0.03)	0.82 (0.02)
refinement of object	0.48 (0.07)	0.51 (0.06)
sub action	0.53 (0.09)	0.59 (0.06)
sub argument of action	0.20 (0.09)	0.28 (0.12)
sub priority	0.72 (0.13)	0.73 (0.10)
sub role	0.69 (0.14)	0.83 (0.08)
sub object	0.48 (0.12)	0.50 (0.06)
sub refinement of object	0.11 (0.13)	0.49 (0.21)
Micro-F_1	**0.72** (0.02)	**0.75** (0.02)
Macro-F_1	**0.60** (0.03)	**0.68** (0.03)

6. Conclusion

With the high complexity of programs, it is difficult for developers to understand users' ambiguous and incomplete expressions for software requirements. We solved this problem by using NLP or SA with pre-defined semantic categories. Given a small, domain-specific text dataset, suitable text preprocessing and feature engineering with elaborate syntactic structures were constructed for our classifier. We carefully designed syntactic features by using dependency and constituency parsers. Among candidate machine learning models, the SVM with kernel was the best model as a single model in terms of the micro-averaged F_1 score (Micro-F_1), whereas the VC was the best in terms of the macro-averaged F_1 score (Macro-F_1). From several experiments, we conclude that syntactic features are more important than bag-of-n-gram features, and that the performance can be improved when we have a larger dataset. Furthermore, we found that our proposed model performed better than the previous model, REaCT. Because the main problem was that the dataset is not enough in size, we believe that if we could collect more data, bag-of-n-gram features and semantic features, such as word embedding, would become more useful.

Acknowledgments

This paper is a revised and expanded version of a paper entitled "Semantic Annotation of Software Requirements with Language Frame" presented at the 14th 2017 International Interdisciplinary Workshop, Daejeon University, on December 21-23, 2017. This research was supported by the Korea Institute of Science and Technology Information (KISTI). Also, this work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre, On-The-Fly Computing''(SFB 901). We specially thank to Frederik Simon Bäumer for his support.

References

[1] A. Hoonlor, B. K. Szymanski, M. J. Zaki and J. Thompson, "An evolution of computer science research", RPI Technical Report 12-01, Rensselaer Polytechnic Institute, Troy, NY, **(2012)**.
[2] I. Sommerville, "Software Engineering: (8th Edition) (International Computer Science)", Addison-Wesley Longman Publishing Co. Inc. Boston, MA, USA, **(2006)**.
[3] M. Dollmann and M. Geierhos, "On- and Off-Topic Classification and Semantic Annotation of User-Generated Software Requirements", Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, S. 1807-1816, Austin, TX, USA, 1. – 5, **(2016)**.
[4] L. Chiticariu, Y. Li and F. R. Reiss, "Rule-based information extraction is dead! long live rule-based information extraction systems!", EMNLP. No. October, **(2013)**.
[5] J. D. Palmer, Y. Liang and L. Want, "Classification as an approach to requirements analysis", Advances in Classification Research Online 1.1, **(1990)**, pp. 131-138.
[6] Liang and Yiqing, "Software requirements classification: definition, approaches, and applications", **(1992)**.
[7] D. Jurafsky and J. H. Martin, "Speech and Language Processing", 3rd. draft edition, **(2015)**.
[8] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning", Proceedings of the 25th international conference on Machine learning. ACM, **(2008)**.
[9] R. B. Lees and N. Chomsky, "Syntactic structures", Language 33.3 Part 1, **(1957)**, pp. 375-408.
[10] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection", Journal of machine learning research 3.Mar, **(2003)**, pp. 1157-1182.
[11] M.-C. De Marneffe and C. D. Manning, "Stanford typed dependencies manual", Technical report, Stanford University, **(2008)**.
[12] H. Yang and C. Zong, "Learning Generalized Features for Semantic Role Labeling", ACM Transactions on Asian and Low-Resource Language Information Processing 15.4, **(2016)**, p. 28.
[13] V. Punyakanok, D. Roth and W.-T. Yih. "The importance of syntactic parsing and inference in semantic role labeling", Computational Linguistics 34.2, **(2008)**, pp. 257-287.
[14] N. Xue and M. Palmer, "Calibrating Features for Semantic Role Labeling", EMNLP, **(2004)**.
[15] C. D. Manning, P. Raghavan and H. Schutze, "Introduction to Information Retrieval", Cambridge, UK: Cambridge University Press, **(2008)**.

Authors

Yeongsu Kim, he is a MS student at University of Science and Technoloy (UST), Korea and Korea Institute of Science and Technology Information (KISTI). His research interests include text mining, natural language processing using machine learning methods.

Seungwoo Lee, he works as a principal researcher at Korea Institute of Science and Technology Information (KISTI), Korea. He received his M.S. and Ph.D. degrees in Computer Science and Engineering from POSTECH, Korea in 1999 and 2005. Previously, He studied and developed an information retrieval engine and a named entity recognizer based on Natural Language Processing (NLP) technologies. And recently he has researched and developed Semantic Web-related tools such as a triple store, an inference engine named OntoReasoner, and a SPARQL processor. He has participated in many Semantic Web-related projects and published as W3C use cases. His current research interest is horizon scanning and issue detection based on text mining and big data analysis.

Markus Dollmann, he is a PhD student at Paderborn University, Germany. His research focus is on semantic information processing. He is especially interested in information extraction methods with and without using machine learning techniques, data cleansing and data harmonization approaches as well as information retrieval systems. During his work for the Collaborative Research Center "On-The-Fly Computing", he worked on automated semantic annotation in user-generated software requirements.

Michaela Geierhos, she heads the chair of digital humanities at Paderborn University, Germany. Between 2013 and 2017, she was assistent professor of business information systems, especially semantic information processing in Paderborn, Germany. After completing an graduate degree in computational linguistics, computer science and phonetics at LMU Munich, she worked as a research associate at LMU's Center for Information and Language Processing between 2006 and 2012. In 2010 she completed her doctoral degree in computational linguistics, for which LMU Munich awarded her a summa cum laude honor. She has received several awards for her contributions to research and teaching. For instance, she was named Professor of the Year 2013 in the category of engineering and computer science and received the Young Researcher's Award from the German Society for Applied Linguistics in 2012. Her research focus is on natural language processing.