Ambiguous Software Requirement Specification Detection: An Automated Approach

Mohd Hafeez Osman
Technical University of Munich, Germany
University Putra Malaysia, Malaysia
hafeez.osman@tum.de

Mohd Firdaus Zaharin
Ministry of Education, Malaysia
firdaus.zaharin@moe.gov.my

ABSTRACT

Software requirement specification (SRS) document is the most crucial document in software development process. All subsequent steps in software development are influenced by the requirements. This implies that the quality of SRS influences the quality of the software product. However, issues in requirement, such as ambiguities or incomplete specification may lead to misinterpretation of requirements which consequently, higher the risk of time and cost overrun of the project. Finding defects in the initial phase is crucial since the defect that found late is more expensive than if it was found early. Thus, the requirement should be tested before moving to other development phases. This study describes an automated approach for detecting ambiguous software requirement specification. To this end, we propose the combination of text mining and machine learning. Since the dataset is derived from Malaysian industrial SRSs and the respondents' that evaluated our result are from Malaysia, we focus this study on Malaysian context for a pragmatic reason. We used text mining for feature extraction and for preparing the training set. Based on this training set, the method 'learns' to classify ambiguous and non-ambiguous requirement specification. In this paper, we study a set of nine classification algorithms from the machine learning community and evaluate which algorithms perform best to detect ambiguous software requirement specification. We take a step forward to develop a working prototype to evaluate the reliability of our approach. The validation of the working prototype shows that the classification result is reasonably acceptable. Even though this study is an experimental benchmark, we optimist that the result of this study may contribute to enhance the quality of SRS and as well as assisting the requirement testing or review.

KEYWORDS

Software Engineering, Requirement Engineering, Machine Learning, Text Mining

ACM Reference Format:
Mohd Hafeez Osman and Mohd Firdaus Zaharin. 2018. Ambiguous Software Requirement Specification Detection: An Automated Approach. In *Proceedings of International Workshop of Requirement Engineering and Testing (RET2018).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.475/123_4

RET2018, June 2018, Gothenburg, Sweden

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

Software requirement is the foundation of software development. Hence, high-quality software requirements specification (SRS) may increase the possibility of high software quality. It is just like the term "garbage in, garbage out" that has been used in software programming which means "If there is a logical error in software, or incorrect data are entered, the result will probably be either a wrong answer or a system crash" [10]. For companies that outsource their software development, software requirements document is the most crucial communication document that specifies the stakeholder's vision and needs of software to be developed. Hence, a good software requirements document is needed to ensure the software developers understand the stakeholder needs.

Research Problem. Issues in software requirements, such as ambiguities or incomplete requirements specification can lead to time and cost overrun in a project [13]. A problem in a software requirement specification need to be detected as early as possible since the problem that found late in the project is more expensive than if it was found early [8]. Some of the issues in requirements specification can be manually detected by the requirements engineers and some are not possible. For example, the requirements specification that ambiguous, unclear and incomplete can be easily detected by the requirement engineer but the requirements that related to the domain knowledge is difficult to be detected. Consequently, an approach that offers requirements engineers rapid detection to a possible defect in specification could contribute valuable feedback [12].

Software requirements defects detection based on requirements template (also known as boilerplate) is feasible based on the work by Arora et. Al [6],5. As the baseline, Arora et. al used the requirement templates from the ISO/IEC/IEEE 29148 [1] and the template proposed by Pohl and Rupp [24]. However, since requirements in the industry are nearly exclusively written in Natural Language, it is hard to detect the issues in requirements because a natural language has no formal semantics. In the Malaysia context, most of the Industrial SRSs are written in Malay. A lot of research has been conducted to solve this problem are focused on English. There are little works focused on Malay. Furthermore, the boilerplates for formulating requirement are not commonly used in Malaysia that make the requirements testing or review much more difficult. While most of the work used Natural Language Processing (NLP) to detect requirements defect, another possible technique for detection of requirements detection is by using text mining. Text mining generally refers to the process of extracting interesting and non-trivial patterns or knowledge from unstructured text documents [19]. Since the requirements are in the form of unstructured

text, text mining is a suitable technique for detecting ambiguous requirement specifications.

Goal. The main objective of this work is to come out with an automated approach on detecting the defect in software requirement specification. In this paper, we aim at formulating an automated approach to detect ambiguous software requirement specification. To this end, we propose a combination of text mining and supervised machine learning. We focus on detecting ambiguous SRS written in Malays since the dataset is derived from Malaysian Industrial software development and the respondents for validating our proposed solution are from Malaysia.

The dataset consists of occurrence of feature-words. This dataset is originated from 180 requirement specifications (gathered from 4 Malaysia Industrial SRSs). The 180 requirement specification have been tagged or labeled manually (ambiguous or non-ambiguous). Supervised machine learning technique is used learn and classify the ambiguous requirements. We explore nine (9) classification algorithms to find the suitable classification algorithms for our purpose. The classification algorithms are OneR, Naive Bayes, Logistic Regression, Nearest Neighbour (k-NN), Decision Table, Decision Stump, J48, Random Forest and Random Tree. We develop a working prototype to evaluate the reliability of the result and our overall approach. The working tool is validated by ten (10) requirement engineer/system analyst.

Contribution. The contributions of this paper are the following:

- Formulation of feature-words based on SRS document.
- Exploring predictive power of feature-words
- Evaluation of nine classification algorithms for detecting ambiguous requirements.
- An automated tool-supported approach for detecting ambiguous software requirements.

This paper is structured as follows. Section 2 discusses the related work. Section 3 describes the research questions and Section 3 explains the approach. We present the results and findings in Section 5. Discussion and Future Work in Section 7. This is followed with conclusion in Section 8.

2 RELATED WORK

Various works have been done on software requirements quality assurance. There are also works focusing on the classification of quality characteristics (e.g. [9]) and others develop a comprehensive checklist [4], [7] and [26]].

Arora et. al [5] proposed an automated solution and tool-supported [6] approach for checking conformance to requirements templates. The work is based on two (2) requirement templates which are Rupp's templates [8] and Easy Approach to Requirement Templates (EARS) [20]. The approaches build on a natural language processing technique, known as text chunking.

Femmer et. al [12] proposed an approach for smelling bad requirement based on requirement quality criteria listed at the ISO 29148 - requirement engineering standards [1]. This work aimed at rapidly detecting requirements that violate the Requirement Engineering principles. They also develop a tool to detect the requirements principles violation by using part-of-speech (POS) tagging, morphological analysis and dictionaries.

Alshazly et. al. [3] concerned with detecting defects in software engineering specification based on defect taxonomies. They proposed a taxonomy focusing on the defects in the requirements phase and added correlations between the defects and the causes of their occurrences to guarantee the quality of SRS documents. The taxonomy intended to help in training the software engineer, SRS writers, support creating accurate requirements and efficient requirement defect prevention and requirement defect detection.

Haron et. al. [15] formulated a conceptual model on managing lexical ambiguity to reduce the possibility of misinterpretation errors in Malay sentences by identifying potential Malay vague words. Prior to this work [16], they performed a research to identify a list of potential ambiguous word in Malay language that intended to assist SRS writers or requirement engineers to avoid using the vague words while documenting SRS. This ongoing study has collected 120 potential ambiguous Malay words which can potentially be used as the reference in developing SRS.

There are also work on improving software requirements specification by using NLP and text mining such as the work by Sateli et. al. [25]. The summary for the recent works are the following:

a. Most of the works for detecting defect in requirements specification refers to several requirement structures such as Rupp's, EARS, IEEE 830 and ISO/IEC/IEEE 29148. From our observation, not all requirements following the requirement structure but still the requirement are still understandable or in fair quality requirement. It is important to have a technique that detect requirements defect based on truly unstructured text.
b. It is difficult to validate the requirements defect detection based on the aforementioned requirement templates since case studies that follow the requirement structure are limited.
c. Requirement specification is highly related to the underlying language used to express the requirement. Most of the works focus on English as the basic language. The list of ambiguous words provided by Haron et. al. [15],[16] is beneficial to be used in our research which focuses on Malaysian Context.

3 RESEARCH QUESTIONS

In the context of the research problems specified in section 1, this paper intended to answer the following research questions:

RQ1. What are the influential words in classifying ambiguous software requirements? We explore the predictive (classification) power of each words that exist in software requirement specifications which are used in this study. We also evaluate the predictive power of words that are suggested by Haron et. al[17] and are mentioned in Berry et. al. [8].

RQ2. What are suitable classification algorithms in classifying ambiguous software requirement? The candidate classification algorithms are evaluated to find the suitable algorithm(s) in classifying ambiguous software requirement.

RQ3. Which set of words produce the best classification performance? We explore how the performance of the classification algorithm is influenced by partitioning the words in different sets.

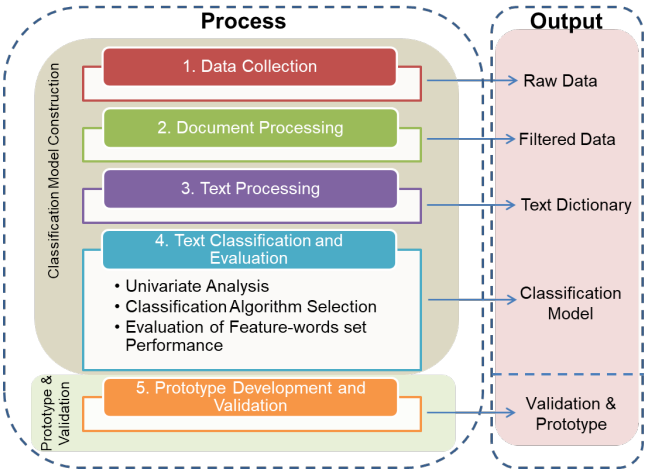

Figure 1: Overall Approach

4 APPROACH

This section describes the overall approach (as illustrated in figure 1) that consist of data collection, data preprocessing, text processing, text classification, prototype development and validation.

4.1 Data Collection

The main data for this work is the SRS documents. As mentioned in section 1, this study is an early work of automated detection on ambiguous software requirement for SRS written in Malay. Thus, we have selected four (4) SRS documents from four (4) separated (Malaysian) software development projects. Out of these four (4) documents, we extracted 180 software requirement specifications that we believe are suitable to be in the dataset.

4.2 Document Preprocessing

In Document Preprocessing, there are two (2) major activities involved which are data labeling and data cleansing or filtering.

a. Data Labeling
In data labeling activity, we manually classify each extracted requirement specification into (i) ambiguous or Y and, (ii) non-ambiguous or N. This labels are crucial for supervised machine learning purposes.

b. Data Cleansing
Data Cleansing refers to the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data [27]. This activity will clean or scrub the unstructured text (software requirement specification) that consists of stop word removal, stemming and word filtering. Since the library for stop word removal and stemming for Malay language is not available, this activity is conducted manually. We also convert all short forms words into formal words (e.g. "yg" to yang).

4.3 Text Processing

The expected output of this activity is a text dictionary that consist of classification features and labels. We consider all words that are exist in the selected requirement specifications as our features (called feature-words). In total, there are 405 feature-words gathered from 180 requirement specifications. The text dictionary is created based on the following steps:

(1) Calculate all number of occurrence for each feature-words in each software requirements.
(2) Integrate the label and the software requirements.

This text processing activity is supported by RapidMiner [2]. An example of text dictionary is illustrated in Table 1.

4.4 Text Classification

The text classification process aims at (i) performing univariate analysis to measure the predictive power of each feature-word, (ii) selecting the suitable classification algorithms for building a classification model to classify ambiguous and non-ambiguous requirement and, (iii) evaluating different sets of feature-word. The explanation of these tasks are the following:

Univariate analysis

To measure predictive power of predictors (feature-words in our case), we used the information gain with respect to the class [33]. Univariate predictive power means measuring how influential a single feature-word is in prediction or classification performance. The results of this algorithm are normally used to select the most suitable predictor for prediction purposes. Nevertheless, in our study we did not use it for predictor selection, but for an exploratory analysis of the usefulness of various feature-words. We compare the feature-words that we found in our study with the ambiguous words that are suggested by Haron et. al. [15] and the words mentioned in Berry et. al. [8].

Classification algorithm selection

This process is divided into several task: (i) classification algorithms candidate selection; (ii) classification model construction and (iii) classification model evaluation. The detail explanations are the following:

i. *Classification algorithms candidate selection*
Prior to making a selection of the classification algorithms, we ran exploratory experiments on a wider range of supervised machine learning algorithms. We do not expect that there will be a single silver bullet algorithm that will outperform all others across our dataset. Also, we are not just interested in a single algorithm that scores a top result on a given problem, but are looking for sets of classification algorithms that produce a reasonable result for the dataset. In this way, we will open the possibility of mixing the classification algorithms for better classification performance. As mentioned above, we desired to explore a diverse set of classification algorithms representative for different approaches. For example, decision trees, stumps, tables and random trees or forests all divide the input space up in disjoint smaller sub spaces and make a prediction based on occurrence of positive classes in those sub spaces. K-NN are

Table 1: Text Dictionary Example

	yang	sistem	akan	maklumat	dan	untuk	telah	mohon	guna	boleh	proses	..(405 words)	Label
Req.1	0	0	0	0	0	1	0	0	0	1	0	..	N
Req.2	1	0	0	1	0	0	0	0	0	1	0	..	Y
Req.3	2	1	0	0	0	1	0	0	0	0	0	..	Y
Req.4	1	0	1	0	2	3	0	0	1	0	2	..	Y
Req.5	2	0	1	0	1	0	0	0	1	0	0	..	Y
Req.6	1	0	0	1	1	0	0	0	0	2	1	..	Y
Req.7	0	0	1	0	0	1	0	0	0	0	1	..	N
Req.8	4	0	1	0	0	1	0	0	0	1	0	..	Y
Req.9	1	0	0	0	0	0	0	0	0	0	0	..	N

similar local approaches, but the sub spaces here are overlapping. In contrast, logistic regression and Naive Bayes model parameters are estimated based on potentially large number of instances and can thus be seen as more global models [21]. On the other hand, OneR generates one-level decision tree expressed in the form of a set of rules that all test one particular attribute [14]. More explanation about these algorithms can be found at [14] and [22].

ii. *Classification model construction*
This task is supported by WEKA [14] (tool). We used default configuration suggested by WEKA for the training activity for each classification algorithms. For every training and test activity for the classification algorithm, we used 10-fold cross validation, it means that we use only 10% of the data for testing and 90% for training. To further improve reliability, we ran each experiment 10 times using different randomization.

iii. *Classification model evaluation*
The classification algorithms are considered to be suitable for our purpose based on its classification performance. Since our dataset is reasonably balance, we evaluate the classification performance based on three measurements: (i) accuracy (percentage of correct classification), (ii) precision () and, (iii) recall. At first, we evaluated the classification algorithms performance against accuracy of ZeroR (the baseline). ZeroR predict the majority class (if nominal) or the average value (if numeric) [14]. This classifier does not take feature as the predictor and present the percentage of probability or random guessing. Accuracy value below ZeroR shows that the model (created by the classification algorithm) does not make any significant improvement in classifying the ambiguous requirement compared to random guessing. Hence, the classification algorithm accuracy value should better higher than ZeroR. Then, we compare the classification algorithm based on precision and recall. Precision and recall are common in information retrieval for evaluating classification performance [11]. If needed, we evaluate the F-Measure (balance between precision and recall) of the classification algorithms performance.

Evaluation of feature-words set performance

The univariate analysis is intended to show the predictive performance for individual feature-words. However, this analysis does not show the predictive performance for set of features. To evaluate the performance sets of feature-words, we created additional three (3) datasets: (i) use only words suggested by Haron et. al as set of feature-words, (ii) use only words mentioned in Berry et. al. as feature-words and (iii) combine both datasets from (i) and (ii). We evaluated the performance of all datasets against all classification algorithm candidates.

4.5 Prototype Development and Validation

In this study, we aim at formulating an automated approach to assist requirement engineers to detect ambiguous software requirement specification. We build an interactive prototype tool to classify ambiguous and non-ambiguous requirement specification based on the evaluated classification model. Based on this prototype, we validate our (initial) approach by conducting a survey.

5 RESULT AND FINDINGS

This section describes i) evaluation on predictive power of feature-words, (ii) evaluation on classification algorithms performance and, (iii) analysis of datasets. Each of the following subsection answers the Research Questions that have been mentioned in section 3.

5.1 RQ1: Feature-words evaluation

Attribute Evaluator analysis from WEKA (tool) is used to evaluate the influence of each feature-words in classifying ambiguous and non-ambiguous requirement specification. This evaluation produces a score from 0 to 1 for every feature-words. A value closer to 1 means strong influence. We consider a predictor as influential when the Information Gain score > 0. Out of 405 feature-words that are extracted from 180 software requirement specification, we found that 94 feature-words have Information Gain score > 0. These feature-words have the score ranging from 0.3010 to 0.0205.

Table 2 shows the list of 30 most influential feature-words. Based on our best knowledge and experience, these words are common words that can be found in SRS documents. Only several words in the list (Table 2) are domain related words such as dana ("*fund*"), majikan ("*employer*") and emel ("*e-mail*"). Which means, the domain related words have little influence in classifying the ambiguous and non-ambiguous requirements. We take a step further to compare the list of influential feature-words with the list of ambiguous words that were suggested by Haron et.al. [17] and were mentioned in Berry et. al [8]. We matched the words that were suggested/mentioned in both study with our feature-words. The Information Gain value for these words are presented in Table 3

for Haron et. al and Table 4 for Berry et. al. We discovered that several words that have been listed by both study influence the classification performance. From this result, we decided to use the list of words by both study as our permanent feature-words in our prototype.

Table 2: List of 30 influential words based on InfoGainAttrEval Score

No	word	InfoGain score	No	word	InfoGain score
1	sistem	0.3010	16	jika	0.0747
2	papar	0.2855	17	buat	0.0662
3	dan	0.2371	18	semak	0.0662
4	akan	0.1320	19	tidak	0.0662
5	tersebut	0.1095	20	pilih	0.0636
6	maklumat	0.1090	21	yang	0.0620
7	hendaklah	0.1036	22	ahli	0.0597
8	dalam	0.1033	23	dari	0.0586
9	dana	0.1023	24	tempoh	0.0579
10	majikan	0.0861	25	telah	0.0579
11	projek	0.0841	26	emel	0.0579
12	kepada	0.0838	27	melalui	0.0579
13	proses	0.0819	28	baru	0.0579
14	boleh	0.0809	29	juga	0.0524
15	senarai	0.0762	30	sekiranya	0.0524

Table 3: Preditive power of feature-words by Haron et. al.

Word	Similar feature-word	InfoGain score
menghantar	hantar	**0.0310**
mengeluarkan	dikeluarkan	0.0000
menerima	terima	**0.0416**
mengemaskini	kemaskini	0.0000
maklumat	maklumat	**0.1090**
meluluskan	lulus	0.0000
notifikasi	notifikasi	0.0000
untuk	untuk	**0.0420**
sekiranya	sekiranya	**0.0524**
khidmat	perkhidmatan	0.0000
dengan	dengan	**0.0318**
permohonan	mohon	0.0000
operasi	operasi	0.0000
pengguna	*not available*	*not available*

5.2 RQ2: Evaluation on Classification Algorithms Performance

As mentioned in section 4, we evaluate the classification algorithms performance based on (i) accuracy, (ii) precision and, (iii) recall. First, we evaluate the classification algorithms performance by comparing the classifications algorithms with ZeroR score. The ZeroR accuracy score is 53%. As illustrated in Table 5, all candidate classification performed better than ZeroR which means that all classification algorithm present significant improvements compare to random guessing.

Table 4: Preditive power of feature-words by Berry et. al.

Word*	Similar feature-word	InfoGain Score
semua	semua	0.0000
setiap	setiap	**0.0222**
sahaja	sahaja	0.0000
juga	juga	**0.0524**
walaupun	walaubagaimanapun	0.0000
atau	ataupun	**0.0257**
beberapa	*not available*	*not available*
bahawa	bahawa	0.0000
yang	yang	**0.0620**

* these words were translated to Malay

Then, we compare the classification algorithms with OneR. OneR is the simplest classification algorithm that takes only one feature for prediction or classification. This evaluation is done because according to Holte[18], there is a possibility that a simple algorithm works well in a dataset. If the performance of using complex classification algorithms are not much different or same with the simplest algorithms, it is better to use simple classification algorithms for better performance (in terms of classification speed). The result shows that OneR performs better than k-NN and Decision Stump while Decision Table score the same result. Hence, k-NN, Decision Stump and Decision Table are considered not suitable as classification algorithms for our purpose.

We further our investigation by evaluating classification performance based on precision and recall score. Again, OneR is used as our benchmark. The result (Table 5) shows that several classification algorithms perform well in recall but not for precision. For example, naive bayes score high for recall (0.95) but low for precision (0.73). To get a better picture on these classification performance, we look at F-Measure i.e. a score that consider both precision and recall. The result shows that Naive Bayes, Logistic Regression, J48, Random Forest and Random Tree are suitable classification algorithms for our purpose. Out of these five (5) classification algorithms, Random Forest is the most suitable classification algorithms based on all the evaluated scores.

Table 5: Classification Algorithms Performance

	Accuracy	Precision	Recall	F-Measure
OneR	78.06	0.80	0.74	0.76
Naive Bayes	80.22	0.73	0.95	0.82
Logistic Reg.	80.94	0.78	0.86	0.81
k-NN	71.89	0.64	0.94	0.76
Decision Table	78.06	0.74	0.84	0.78
Decision Stump	77.17	0.69	0.95	0.80
J48	82.67	0.83	0.82	0.81
Random Forest	89.67	0.90	0.88	0.89
Random Tree	80.89	0.78	0.85	0.81

Table 6: Classification Performance (accuracy) based on Dataset

	AllData	*DataH	*DataB	*DataHB
OneR	78.06	65.78	60.78	65.78
Naive Bayes	80.22	65.33	58.11	67.22
Logistic Reg.	80.94	67.83	54.72	66.67
k-NN	71.89	67.89	58.22	69.83
Decision Table	78.06	62.00	56.72	63.28
Decision Stump	77.17	48.89	57.78	52.33
J48	82.67	71.17	57.83	70.89
Random Forest	89.67	70.44	59.06	76.56
Random Tree	80.89	68.22	59.00	71.39

* Data_H used feature-words suggested by Haron et. al [17]; Data_B used feature-words mentioned in Berry et. al. [8] ; and DataHB is the combination of DataH and DataB

5.3 RQ3: Analysis on Dataset

The accuracy score for all datasets are illustrated in Table 5. This table shows that by using all (405) feature-words, the classification algorithms produce the best results. The other datasets score much lower than using all words as feature-words. This result indicates that it is not enough to only used the words that are suggested by Haron et. al. [17] and words that are mentioned in Berry et. al. [8] to classify ambiguity and non-ambiguity requirements.

6 PROTOTYPE AND VALIDATION

This section describes the development of the prototype and the validation of the prototype.

6.1 Prototype Development

The main objective of this tool is to assist the requirement engineer to develop a good quality requirement specification. This tool displays several useful information on how to formulate a good software requirement specification based on ISO/IEC/IEEE 29148 and the information from International Requirements Engineering Board (IREB). As for now, this tool offers a functionality to detect ambiguous and non-ambiguous requirement specification. A brief description about this functionality is the following:

- Random Forest is used as the classification algorithm
- 180 requirement specifications are used as the learning dataset
- The feature-words shall be dynamically chosen by the user

Figure 2 shows the main page of the tool that display the information about a good requirement and a function to detect ambiguous requirement specification. The user need to provide the requirement specification as the input. Then, the tool classifies the requirement by using the classification model, then presents the classification result (see Figure 3) as well as the information about the words (words occurrence information). This tool is developed using PHP, AJAX, Python and JQuery. We used scikit-learn[23] library for executing the machine learning task.

6.2 Validation

In this study, we desired to provide a pragmatic solution to this problem. Although this study is still in an initial stage, we need to

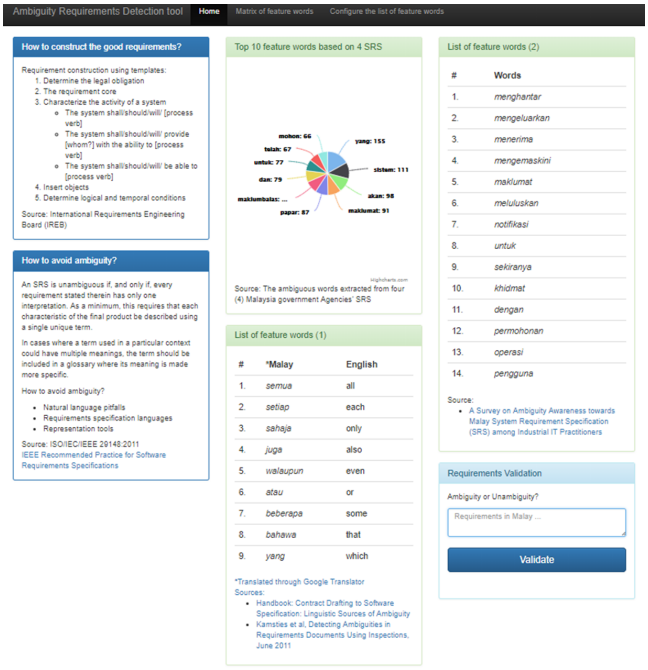

Figure 2: Tool's Main Page

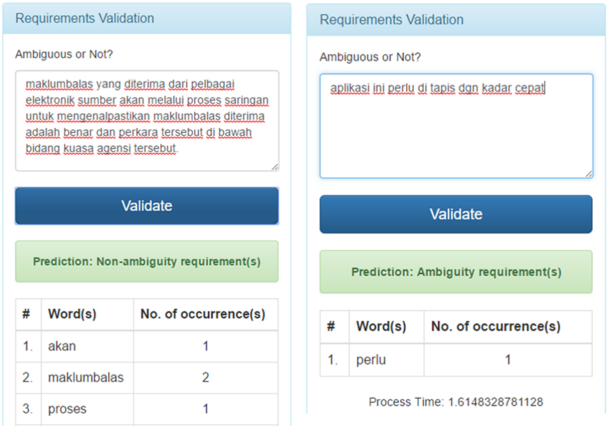

Figure 3: Tool's Classification Results

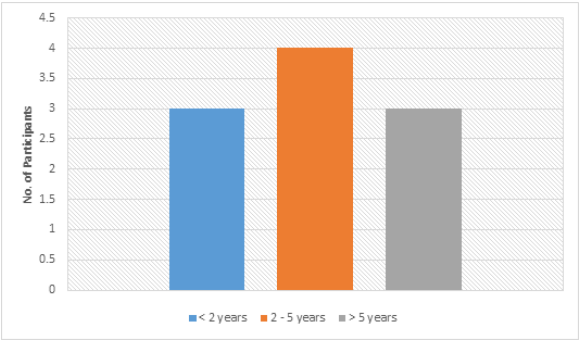

Figure 4: Respondents' Background

observe the practicality of this tool in helping requirement engineer writing a good requirement specification. Thus, we conducted a simple survey to validate the prototype.

This survey is conducted to measure certain attributes based on experts' opinion and review. Those attributes are the Information Quality and the Usefulness of the Tool. The result of this survey is important to indicate the acceptance, satisfaction and usefulness of prototype tool from the experts' view. The respondents were required to use the prototype and then, answer several questions. The format (Likert scale) of a six-level item is used as answer options in this questionnaire, i.e. *strongly disagree*, *moderately disagree*, *disagree*, *agree*, *moderately agree* and *strongly agree*.

Respondents' Background

Overall, we have selected ten (10) respondents that worked as Requirement Engineer and System Analyst to involve in this activity. Out of 10 respondents, three (3) respondents have less than 2 years of experience, four (4) respondents have 2 to 5 years of experience and three (3) respondents have more than 5 years of experience (figure 4).

Information Quality

To assess the quality of information presented in the tool, the respondents were required to answer the following questions:

a. *The result of requirement ambiguity detection is accurate?*
The result shows that none of the respondent disagree that the result of the tool is inaccurate. Seven (7) of the respondents answered *moderately agree*, two (2) of the respondents answered *strongly agree* while only one (1) answered *agree*. This shows that from the respondent's perspective, the result of the ambiguous requirement specification detection is moderately accepted.
b. *The tool provides me with sufficient information?*
In general, we can summarize that the respondents slightly agree that the tools provide sufficient information. From the result, four (4) of the respondents answered *agree*, five (5) answered *moderately agree* while only one (1) respondent answered *strongly agree*. Perhaps, this is due to the prototype only give the classification result and the words that are used but not the information on what kind of word that are contribute to ambiguous software requirement.

Usefulness of the Tool

To assess the usefulness of the tool, the respondents were required to answered two (2) question:

a. *The tool increases productivity in formulating Software Requirement Specification (SRS)?*
Most of the respondents *moderately agree* that the tool increase the productivity in formulating SRS. Three (3) of the respondents answered *strongly agree* while only one respondent answered *agree*.
b. *This tool improves the Requirement Engineer/ System Analyst Performance in preparing SRS?*
Five (5) respondents answered that they are *moderately agree*, while four (4) respondents answered that they are *strongly agree*. This shows that the tool can be used as a learning material for Requirement Engineers / System Analysts to improve their SRS preparation skills.

7 DISCUSSION AND FUTURE WORK

This section presents our discussion on the dataset, classification model construction, prototype tool and threats to validity.

7.1 Dataset

As for now, we have processed four (4) SRS (written) and selected 180 requirement specifications. We are aware that we have a little coverage on SRS. Furthermore, using all words as feature-words is difficult when the data is much bigger. Nevertheless, this study was considered as an early experimental benchmark, we intended to observe the possibility of using text mining and supervised machine learning to solve issues related to ambiguous requirement specification. We see a number of ways to extend this study such as increasing the dataset, improving the ground truth (ambiguous/non-ambiguous label), formulating more feature words (e.g. number of words per specification, formulating weight for common ambiguous word, and expend the ambiguous words that are mentioned by Haron et. al. [15]).

7.2 Classification Model Construction

Generally, this study is expected to find the suitable algorithms to classify ambiguous requirement based on our dataset. This study has shown that Naive Bayes, Logistic Regression, J48, Random Forest and Random Tree are suitable to be used as classification algorithms for our dataset. Based on the classification score, Random Forest shows the best performance in terms of precision and recall. Hence, this classification algorithm is used in the prototype tool. There is a possibility to enhance the performance of Random Forest by reconfiguring the parameter. Also, combination with other classification algorithms can be another alternative.

7.3 Prototype Tool

The purpose of developing the prototype is to evaluate the classification model with the input from the requirement engineer or software analyst. We validate this tool to observe the reliability of the tool in classifying ambiguous and non-ambiguous requirement specification. Moreover, we also like to observe on the usefulness of the prototype in helping requirement engineers and system analysts in formulating requirement specification. Although the result is reasonably acceptable, we see a lot of improvement should be done such as presenting the word that probably contribute to ambiguous requirement, increase more dataset, improve feature-words and validate the tool with more requirement engineer.

7.4 Threats to Validity

This subsection describes the threats to validity of this study that consists of threats to internal validity, external validity and construct validity.

Threats to Internal Validity: The stemming of the words in the requirements specification during the data preprocessing was done manually. There is possibility that we unable to find several root words. Another threats to internal validity is the feature-words that are based on English-Malay translation (words by Berry et. al. [8]). These words are translated literally without comparing other words that are also synonym but in different context.

Threats to External Validity: We use 180 requirement specification that are gathered from four (4) SRS. These SRS only represent several system domain and limited pattern on requirement specification formation. Hence, we could not generalize the result of this study for all systems in Malaysia.

Threats to Construct Validity: We use the default parameter to construct a classification model for each classification algorithms candidate. There is a possibility for a classification algorithm to produce a better classification score if the classification algorithms parameter are configured differently. Also, there is a possibility that several other classification algorithms that would perform better classification that we did not covered in this study.

8 CONCLUSIONS

This study aimed to come up with an automated approach to classify ambiguous requirement specification. For a practical reason, we focus on SRS that are written in Malay. The dataset was created based on 180 software requirement specifications that were extracted from four (4) SRS. We evaluated the predictive power of each words and also compared the highly influential feature-words for classifying ambiguous requirements with existing research. We discovered that several ambiguous words suggested by existing research influenced the classification of ambiguous requirement. However, we found that there are also some other words that are highly influential for classifying ambiguous requirements from our dataset.

We have conducted an experiment to find the suitable classification algorithms for our purpose. We discovered that Naive Bayes, Logistic Regression, J48, Random Forest and Random Tree is suitable for our dataset. Random Forest performed the best classification scores compared to all aforementioned algorithms. Based on the experiment, we developed a working prototype to evaluate the result and the reliability of our classification model. The tool was validated by ten (10) requirement engineers/system analyst. The validation showed that the classification result of the tool is reasonably acceptable. Also, the validation result shows that the tool is useful for improving the quality of SRS. Although this study was considered as an early experimental benchmark, we believe that the result of this study provides a significant contribution for improving SRS quality as well as supporting requirement testing or review task.

ACKNOWLEDGMENTS

The authors would like to thank the respondents of this study for their support and commitment.

REFERENCES

[1] 2011. ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering. *ISO/IEC/IEEE 29148:2011(E)* (Dec 2011), 1–94. https://doi.org/10.1109/IEEESTD.2011.6146379

[2] 2017. RapidMiner. https://rapidminer.com/. (2017).

[3] Amira A Alshazly, Ahmed M Elfatatry, and Mohamed S Abougabal. 2014. Detecting defects in software requirements specification. *Alexandria Engineering Journal* 53, 3 (2014), 513–527.

[4] Bente Anda and Dag IK Sjøberg. 2002. Towards an inspection technique for use case models. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. ACM, 127–134.

[5] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. 2015. Automated checking of conformance to requirements templates using natural language processing. *IEEE transactions on Software Engineering* 41, 10 (2015), 944–968.

[6] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer, and Raul Gnaga. 2013. RUBRIC: A flexible tool for automated checking of conformance to requirement boilerplates. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 599–602.

[7] Daniel M Berry, Antonio Bucchiarone, Stefania Gnesi, Giuseppe Lami, and Gianluca Trentanni. 2006. A new quality model for natural language requirements specifications. In *Proceedings of the international workshop on requirements engineering: foundation of software quality (REFSQ)*.

[8] A Bucchiarone, S Gnesi, G Lami, G Trentanni, and DM Berry. 2006. A New Quality Model for Natural Language Requirements Specification. In *Proc. of the 12th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQâĂŹ06), June 2006, Luxembourg, Grand-Duchy of Luxembourg, Essener Informatik Beitrage, ISBN 3-922602-26*, Vol. 6.

[9] Alan Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledeboer, Patricia Reynolds, Pradip Sitaram, et al. 1993. Identifying and measuring quality in a software requirements specification. In *Software Metrics Symposium, 1993. Proceedings., First International*. IEEE, 141–152.

[10] Dictionary.com. 2018. www.Dictionary.com. (2018). Retrieved 5th February 2018 from http:www.dictionary.com

[11] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[12] Henning Femmer, Daniel Méndez Fernández, Elmar Juergens, Michael Klose, Ilona Zimmer, and Jörg Zimmer. 2014. Rapid Requirements Checks with Requirements Smells: Two Case Studies. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE 2014)*. ACM, New York, NY, USA, 10–19. https://doi.org/10.1145/2593812.2593817

[13] Daniel MÃĺndez FernÃąndez and Stefan Wagner. 2015. Naming the pain in requirements engineering: A design for a global family of surveys and first results from Germany. *Information and Software Technology* 57 (2015), 616 – 643. https://doi.org/10.1016/j.infsof.2014.05.008

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. 2009. The WEKA Data Mining Software: An Update. (2009). Issue 1.

[15] Hazlina Haron, Abdul Azim Abdul Ghani, and Hazliza Haron. 2015. A conceptual model to manage lexical ambiguity in Malay textual requirements. *ARPN Journal of Engineering and Applied Sciences* 10, 3 (2015), 1405–1412.

[16] Hazlina Haron and Abdul Azim Abdul Ghani. 2014. A method to identify potential ambiguous Malay words through Ambiguity Attributes mapping: An exploratory Study. *CoRR* abs/1402.6764 (2014). arXiv:1402.6764 http://arxiv.org/abs/1402.6764

[17] Hazlina Haron and Abdul Azim Abdul Ghani. 2015. A Survey on Ambiguity Awareness towards Malay System Requirement Specification (SRS) among Industrial IT Practitioners. *Procedia Computer Science* 72 (2015), 261–268.

[18] Robert C Holte. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine learning* 11, 1 (1993), 63–90.

[19] Ah hwee Tan. 1999. Text Mining: The state of the art and the challenges. In *In Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*. 65–70.

[20] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. 2009. Easy approach to requirements syntax (EARS). In *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*. IEEE, 317–322.

[21] M. H. Osman, M. R. V. Chaudron, and P. v. d. Putten. 2013. An Analysis of Machine Learning Algorithms for Condensing Reverse Engineered Class Diagrams. In *2013 IEEE International Conference on Software Maintenance*. 140–149. https://doi.org/10.1109/ICSM.2013.25

[22] M. H. Osman, M. R. V. Chaudron, P. van der Putten, and T. Ho-Quang. 2014. Condensing reverse engineered class diagrams through class name based abstraction. In *2014 4th World Congress on Information and Communication Technologies (WICT 2014)*. 158–163. https://doi.org/10.1109/WICT.2014.7077321

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2825–2830. http://dl.acm.org/citation.cfm?id=1953048.2078195

[24] Klaus Pohl and Chris Rupp. 2011. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - IREB Compliant* (1st ed.). Rocky Nook.

[25] Bahar Sateli, Elian Angius, Srinivasan Sembakkam Rajivelu, and René Witte. 2012. Can text mining assistants help to improve requirements specifications. *Mining Unstructured Data (MUD 2012), Canada* (2012).

[26] Axel Van Lamsweerde. 2009. *Requirements engineering: From system goals to UML models to software*. Vol. 10. Chichester, UK: John Wiley & Sons.

[27] Shaomin Wu. 2013. A review on coarse warranty data and analysis. *Reliability Engineering & System Safety* 114 (2013), 1–11.