AMBIGUITY DETECTION IN REQUIREMENTS CLASSIFICATION TASK USING FINE-TUNED TRANSFORMATION TECHNIQUE

Sadeen Alharbi

Department of Software Engineering, King Saud University, Saudi Arabia

ABSTRACT

The software requirement specification (SRS) document is essential in software development. This document influences all subsequent steps in software development. Nevertheless, requirements problems, such as insufficient or ambiguous specifications, can cause misunderstandings during the requirement analysis stage. This influences testing activities and increases the project's duration and cost overrun risk. This paper represents an intuitive approach to detecting ambiguity in software requirements. The classifiershould learn ambiguous features and characteristics extracted from the text on a training set and try to detect similar characteristics from a testing set. To achieve this, this study experimented with two main approaches. The first approach is feature extraction, which uses the hidden states as features and trains asupport vector machine (SVM) classifier to assess software requirement ambiguity without modifying the pre-trained model. Unfortunately, this approach only identified 68% of the requirement ambiguity. The second approach is training an end-to-end model that updates the parameters of the pre-trained model. This approach enhanced the baseline results by 13%.

KEYWORDS

Requirements classification, NLP, Ambiguity.

1. INTRODUCTION

The software engineering field has widely applied various machine learning (ML) techniques to enhance organizations' efficiency and effectiveness. A recent study (Lorenzoni et al., 2021) revealed that ML models could be employed in different development stages of the software lifecycle, especially in the quality and analytics process. Software quality assurance saves time and money, as it helps developers discover errors and mistakes early in the development process. Therefore, the quality of the software requirement specification (SRS) document is essential in software development. This document influences all subsequent steps. Nevertheless, requirements problems, such as insufficient or ambiguous specifications, can cause misunderstandings during the requirement analysis stage. Research reveals that ML could help develop models that can enhance the quality of requirements classification tasks (Lorenzoni et al., 2021). However, selecting and implementing ML techniques to identify ambiguous requirements is not a trivial task. Ambiguity in natural language requirements has long been recognised as an unavoidable challenge in requirements engineering (RE). As a result, RE researchers have launched a variety of initiatives to address the challenges of ambiguity. There isa significant amount of research on using natural language processing (NLP) to address the classification of requirements in natural language. However, there seems to be less interest from the research community regarding the empirical evaluation of ML techniques for detecting various aspects of ambiguities in requirements. This paper demonstrates findings on techniques

David C. Wyld et al. (Eds): MLDS, IBCOM, CSIA, NLAI - 2022
pp. 47-55, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.122105

that address this gap by experimenting with two primary approaches.

The first approach is feature extraction, which uses the hidden states as features and trains a support vector machine (SVM) classifier to assess software requirement ambiguity without modifying the pre-trained model. The second approach is training an end-to-end model that updates the parameters of the pre-trained model using a finetuned DistilBERT (Sanh et al., 2019),a distilled version of BERT (Bidirectional Encoder Representations from Transformers), a recent technique published by Google. DistilBERT is a pre-trained, smaller, general-purpose language representation model that can be fine-tuned with adequate performance on different tasks. The purpose of this study is to present an alternative and enhanced method to evaluate the ML model of the usual feature extraction method. This study's aims are as follows:

- Describe an ML process model for ML classification and perform related ambiguity requirements classification modelling.
- Empirically evaluate random forest and finetuned DistilBERT for textual classificationin the context of ambiguity requirements using the F1-score measure.

The remainder of the paper is structured as follows: Section 2 describes related work. Section 3 explains requirement dependency, extraction, practical relevance and research questions. Section4 elaborates on our ambiguous classification modelling of the ML process. Data used in this study are detailed in Section 5, and empirical results in Section 6. Section 7 details the implications and limitations of this study before summarising conclusions in Section 8.

2. RELATED WORK

A variety of studies have been conducted to improve the quality of software requirements. For example, researchers in (Davis et al., 1993) focused on quality characteristics classification and designed a complete checklist (Berry et al., 2006). Arora et al. 2015 presented an automatic system that supports a toolkit based on checking the conformation approach applied to requirements templates (Arora et al., 2013). The technique is known as text chunking, as it was developed in NLP. In addition, Femmer et al. 2014 presented a method for tracking poor requirements based on selected criteria on the ISO/IEC/IEEE 29148 for software requirement quality (29148, 2011). This study aims to detect requirements that break the principles of RE (Femmer et al., 2014). They also developed a tool for detecting the violation of requirement principles by utilising a part-of-speech (POS) tagging approach, dictionaries and morphological analysis.

Alshazly et al. 2014 examined ways of detecting faults in SRS documents based on specific fault taxonomies. They suggested a taxonomy that concentrates on the faults in the requirement analysis stage and added correlations between them and the causes of their occurrences to ensure the SRS document's quality.

Haron et al. 2015 developed a conceptual model for addressing lexical ambiguity in Malay sentences to minimise the possibility of misinterpretation errors by specifying probable ambiguous Malay words. Before this work, researchers in 2014 conducted a study to specify a list of potential vouge terms in the Malay language, aiming to help SRS requirement engineers avoid employing ambiguous terms while writing SRS documents.

Several works also concentrate on enhancing SRSs by employing NLP and ML, such as the work by Sateli et al. 2012.

Most works in the literature focus on detecting defects in requirements specification, referring to several requirement templates, such as Rupp's, EARS, IEEE 830 and ISO/IEC/IEEE 29148. However, the current study has revealed that not all requirements follow the requirement templates and can be interpreted in different ways due to their ambiguity. Therefore, it is essential to have a technique to detect requirement ambiguity based on a genuinely unstructured text.

Moreover, validating the requirements for defect detection based on the requirement template is challenging, since case studies that follow the requirement structure are limited. In addition, requirement specifications are closely related to the underlying language used to express the requirement. Therefore, focusing on a specific ambiguous type that occurs at the word level can benefit the research.

3. DATA

This section first explains the data collection and preparation process. Then, it clarifies the ambiguity definition in the requirement classification task. Finally, the section demonstrates of agreement measure applied to ensure the quality of the labelling process.

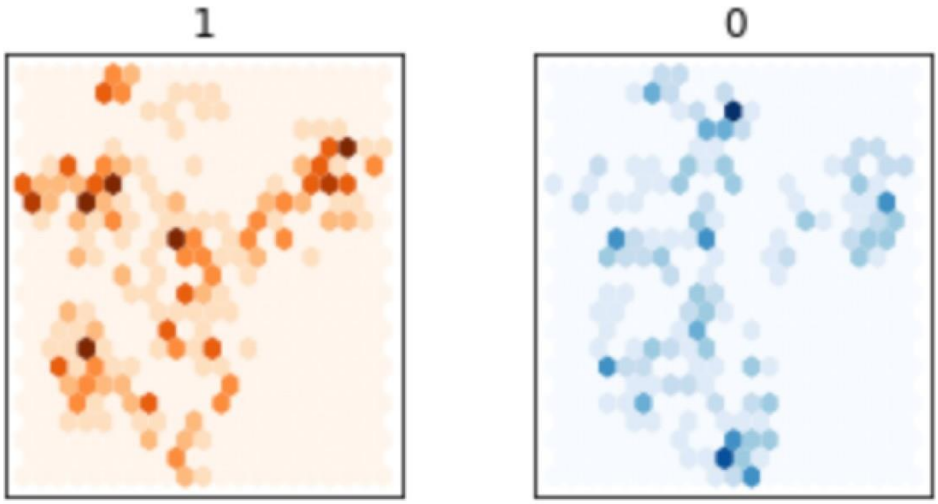

Figure 1: Data demonstration where 1 refers to ambiguous and, 0 refers to unambiguous requirements.

3.1. Collection and Preparation

Table 1: The public available project along with a number of software requirement sentences.

Name of project	No. of SR sentences	Name of project	No. of SR sentences
ABC Paint Project	30	TACHOnet	57
Digital Home System	56	Video Search Engine	24
EStore Project	109	Web Publishing System	81
ReqView Software Requirements	60	Project Bowser	125
IUFA application	31	PDF Split and Merge	27
Total		**600**	

The software requirements sentences were collected from the free database PURE (PUblic Requirements dataset) (Ferrari et al., 2017). A total of 600 software requirements sentences from 10 previous real-world, publicly available, natural language projects were collected. Of these

sentences, 80% were used for training 20% for testing. Table 1 shows the projects and the number of software requirements sentences.

Most software requirements sentences are represented in unstructured and natural language. Consequently, applying the assessment directly to those textual data may lead to low-quality results, increasing the importance of data preparation techniques. The first technique is cleaning the software requirement sentences by removing non-English characters and keeping only the alphabet letters, for example, removing the punctuation and noising characters (, @, $). The second technique is stop word filtering, one of the most widely used pre-processing steps across different NLP applications. Stop words are words that help build ideas but do not have any independent significance, such as conjunctions, articles and prepositions. Stop word filtering eliminates words that will not affect the ambiguity assignment process such that, in the feature extraction phase, the vectorisation of the term will derive only from the assertion that carries meaning and effect in ambiguity detecting.

3.2. Ambiguity Definition in RequirementsClassification

Ambiguity is a complicated and multi-faceted concept (Gervasi and Zowghi, 2010). A better understanding of the nature and source of ambiguity can help in addressing the challenges of requirements by allowing more informed decision making. Ambiguity in natural language arises from different sources, such as a consistent symbolic interpretation, a conceptual problem, or a syntactic or linguistic problem. However, determining ambiguity's root cause is difficult (Gervasi and Zowghi, 2010). Therefore, the researcher provides various ambiguity taxonomies and classifications to study different types of natural language. One considerable classification was provided by Berry; Berry and Kamsties, which taxonomises ambiguity as such:

- Lexical ambiguity occurs at the word level when a word has several possible meanings.
- Structural/syntactic ambiguity occurs at the sentence level when a sentence can be interpreted in more than one way.
- Semantic ambiguity occurs at the sentence level when there is no syntactic or lexical ambiguity, but the sentence predicate logic can lead to several interpretations.
- Language error ambiguity is a grammatically faulty structure that leads to a different meaning due to the error.
- Pragmatic ambiguity concerns the relationship between a sentence's context and its interpretations. Pragmatic ambiguity usually occurs because of the uncertainty of traditional knowledge and human knowledge about context.

This paper only focuses on lexical ambiguity, as this type occurs at the word level and can be written in particular rules. The model follows the rules of avoiding ambiguity mentioned by authors Karl Wiegers and Joy Beatty as follows:

- The sentences must be clear with regard to any adjectives, adverbs and adverbial phrases without any supported by values that describe the level of services—for example, 'small system'.
- The sentences must be clear with regard to any phrases that modify other words in a sentence— for example, 'user-friendly system'.
- The sentences must be clear with regard to vague words, such as 'clear', 'all', 'many', and 'some'.
- The sentences must be clear with regard to any indefinite pronouns that do not refer to a specific person, amount or thing—for example,'any time'.

3.3. Agreement Measure: Cohen's kappa coefficient

The author labelled and judged each sentence in the dataset and assigned a value of one if the sentence was ambiguous and zero if it was unambiguous. Since the author evaluated the training and test set and the data are overlapping, as shown in Figure 1, it is recommended that the processes be verified with evaluations conducted by external experts for a sample of the data. The sample should comprise at least 10% of the test cases.

Kappa is applied in the domains of meta-analysis and content analysis when a researcher needsto determine the accuracy of coding for nominal variables based on raters' agreement. The kappa statistics include classifying N items into C mutually exclusive categories; then, the agreement is calculated. There are variations in the way the kappa coefficient is calculated—for example, the difference in the number of raters and whether the distribution is fixed or free. Fixed marginal kappa is used when the raters must distribute a specific number of items under each category; free marginal kappa does not restrict the number of cases placed into each category (Randolph, 2005). In the present study, the kappa values range from -1 to 1, where -1 means 'complete disagreement' and 1 means 'perfect agreement'. The most common results for the kappa values are shown in Table 2.

Table 2: Interpretation of Kappa (Randolph, 2005).

Values	Interpretation
<0	Less than chance agreement
0.01–0.20	Slight agreement
0.21– 0.40	Fair agreement
0.41–0.60	Moderate agreement
0.61–0.80	Substantial agreement
0.81–0.99	Almost perfect agreement

We performed an inter-judge agreement analysis on 11% of the cases studied in this project. Two external raters worked on the task; the first was the author, and the second was an experienced software engineering employee. These raters compared their classifications, and free marginal kappa was applied, since there was no restriction on the number of cases under each category. Table 4 shows the kappa statistics results and their interpretation.

Table 3: Kappa statistics and their interpretation

Requirement #		Interpretation
1	0.5968	Moderate agreement
2	0.6781	Substantial agreement
3	0.6667	Substantial agreement
4	0.6355	Moderate agreement
5	0.5837	Moderate agreement
6	0.8163	Substantial agreement
Average	0.6566	**Substantial agreement**

Most of the results were either 'moderate agreement' or 'substantial agreement'. In general, the results were acceptable, as the average agreement was 'substantial'.

4. METHODOLOGY

To build the requirement classification system, it is necessary to train a model that can classify requirement sentences into the categories of ambiguous or unambiguous. To do so, this study employed a variant of BERT called DistilBERT (Sanh et al., 2019). The main benefit of this model is that its performance is similar to that of BERT while being smaller and more efficient. Furthermore, applying this model enables the researchers to train a classifier in a short amount of time.

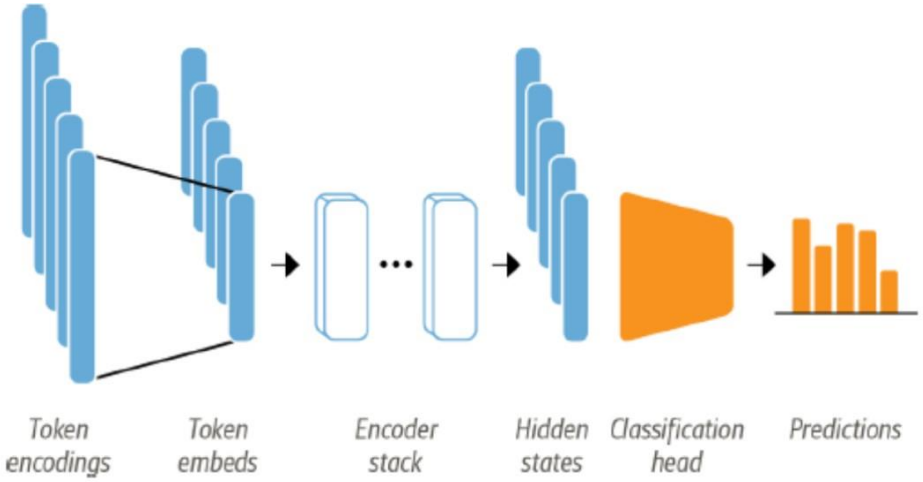

Figure 2: Sequence classification architecture with an encoder-based transformer (Tunstall et al., 2022).

4.1. Tokenizing the Whole Dataset

As a transformer model, DistilBERT cannot accept a string in its original raw form as an input. Instead, transformer models usually suppose that the text has been tokenised into numerical vectors. Tokenisation is when a string is broken down into atomic units. Different tokenisation techniques exist to be adapted, and the best approach of words splitting into sub-units is usually comprehended from the dataset.

The Map() method was applied to tokenise the whole dataset. This method provides a suitable way to use a processing function for each sentence in a dataset. The Map() function applies the tokeniser to a set of sentences. After preparing a Map() as the primary processing function, it is possible to use it across all the splits in the dataset. By default, the Map() method operates on each sentence in the corpus.

4.2. Training a requirements Classifier

According to (Tunstall et al., 2022), pretrained models like DistilBERT are usually applied in a text sequence to predict masked words. However, in the requirement classification task, using language models (LM)s directly is impossible, as some modifications are needed (see Fig. 2).

The first step is accomplished when the token encoding is created. This happens when the text is tokenised and expressed as one-hot vectors. Then, all token encodings are transformed into token embeddings. These token embeddings are vectors inhabiting a lower-dimensional space. Next, token embeddings are handled via the encoder layers to generate a hidden state for each input token. Finally, each hidden state is provided to a layer that predicts the masked input tokensfor the

pretraining objective of language modelling. This replaces the LM layer with a classification layer for the classification task. Then, we have two options to train the model on our requirements classification data: feature extraction and fine tuning.

4.2.1. Transformers as Feature Extraction

Extracting features to build a classifier using a transformer is relatively simple. During the training phase, the hidden states are used as input features and the labels as targets for the classifier, while the weights of the parameters are frozen. Then, different models are trained on the most popular classifiers in binary text classification. The machine learning techniques are logistic regression (LR), support vector machine (SVM) and Knearest neighbours (KNN). The training is applied on the classifiers without modifying the pre-trained model.

4.2.2. Fine-Tuning Transformers

In this approach, the whole end-to-end model is trained, including the update process of pre-trained model parameters. The hidden states are trained as demonstrated in Figure 2, as they are not fixed, as in the approach explained in 4.2.1. First, the pretrained DistilBERT model is used. Then, the researcher logs in to huggingface (Wolf et al., 2019) and utilises the TrainArguments class to define the training parameters.

5. EXPERIMENT

This section presents the experiments on the prepared data demonstrated in Section 3 using the method discussed in Section 4.2.1 and 4.2.2. The following experiments evaluate the system's ability to detect and classify the ambiguity of natural requirement sentences. Results are described using the conventional F-score, which describes the relation between recall and precision as follows:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \qquad (1)$$

where **precision** measures the percentage of correctly classified samples in relation to the total samples, and **recall** is the number of correct positive results divided by the number.

5.1. Baseline experiments

At this stage, the pre-processed dataset now contains all the information required to train a classifier on it. The hidden states were used as input features and the labels as targets. The corresponding arrays were created in the well-known Scikit-learn format (Pedregosa et al., 2011). As mentioned above, the experiment was done for the most popular classifiers in binary text: LR, SVM and KNN. The dataset was split into training and testing sets, where 80% of the datawere used for training and 20% for testing. Table 4 shows that the highest F-score is obtained from the LR classifier; the results of all utilised classifiers are approximate.

Table 4: Comparing between system results in baseline experiments

Classifier #	F-score
Logistic Regression	68%
K-Nearest Neighbors	63%
Support Vector Machine	66%

The results indicate that popular classifiers are able to learn from trained data to detect ambiguityin natural requirement sentences. However, their efficacy is insufficient for companies to apply these classifiers in real-life projects.

5.2. Decoding with Fine-Tuning Transformers

All models were trained for 20 epochs with a batch size of 64 with learning rate 2e−5. All parameters of the model were fine-tuned during training, and no layer was left frozen. Then, the model was evaluated on the test dataset. The ability of the model to detect ambiguity in requirement sentences achieved an F-score of 77%. This represents an enhancement of 13% compared to LR, 22% compared to KNN and 16 The result of this work has not been compared with others, because prior work was more focused on classifying the functional and non-functional requirements rather than detecting ambiguity in natural requirement sentences.

6. CONCLUSIONS

In conclusion, this study introduces a software requirements classification system intended to assist software engineering in writing requirements sentences. The system is designed to detect potentially ambiguous words in software requirements sentences and classify them as ambiguous or not. First, three classification approaches (LR, SVM and KNN) were applied in the system implementation. Then, the system was enhanced using a finetuned transformer technique and successfully classified the ambiguous requirement sentences with an F-score of 77%. This result indicates that the system achieved the research objectives.

While conducting this work, some difficulties were faced during the implementation phase. Detecting ambiguity in software requirements sentences is a new area of research, and few academic resources are available. This limitation creates a challenge to include detecting ambiguous phrases in requirement sentences and requires high-quality rules to detect them.

ACKNOWLEDGEMENTS

This research project was supported by the Deanship of Scientific Research, King Saud University, Saudi Arabia, by a funding through Vice Deanship of Scientific Research Chairs. Also, I would like to thank my student: Saja Alharbi for collecting the data for her project for the year 2021.

REFERENCES

[1] ISO/IEC/IEEE 29148. (2011). “Systems and software engineering—life cycle processes—requirements engineering”.
[2] Amira A Alshazly, AhmedMElfatatry, and Mohamed S Abougabal. 2014. Detecting defects in software requirements specification. Alexandria Engineering Journal, 53(3):513–527.
[3] 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):944–968.

[4] 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, pages 599– 602.

[5] Daniel M Berry. (2007). "Ambiguity in natural language requirements documents". In Monterey Workshop, pages 1–7. Springer.

[6] 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).

[7] Daniel M Berry and Erik Kamsties. (2004). "Ambiguity in requirements specification". In Perspectives on software requirements, pages 7–44. Springer.

[8] 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 [1993] Proceedings First International Software Metrics Symposium, pages 141–152. IEEE.

[9] Henning Femmer, Daniel M.ndez Fern.ndez, Elmar Juergens, Michael Klose, Ilona Zimmer, and J.rg Zimmer. (2014). "Rapid requirements checks with requirements smells: two case studies". In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pages 10–19.

[10] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. (2017). "Pure: A dataset of public requirements documents". In 2017 IEEE 25th International Requirements Engineering Conference (RE), pages 502–505. IEEE.

[11] Vincenzo Gervasi and Didar Zowghi. (2010). "On the role of ambiguity in re". In International Working Conference on Requirements Engineering: Foundation for Software Quality, pages 248–254. Springer.

[12] Hazlina Haron and Abdul Azim Abd Ghani. (2014). "A method to identify potential ambiguous malay words through ambiguity attributes mapping: An exploratory study". arXiv preprint arXiv:1402.6764.

[13] 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):1405–1412.

[14] Giuliano Lorenzoni, Paulo Alencar, Nathalia Nascimento, and Donald Cowan. (2021). "Machine learning model development from a software engineering perspective: A systematic literature review". arXiv preprint arXiv:2102.07574.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. (2011). "Scikit-learn: Machine learning in Python". Journal of Machine Learning Research, 12:2825–2830.

[16] Justus J Randolph. (2005). "Free-marginal multirater kappa (multirater k [free]): An alternative to fleiss' fixed-marginal multirater kappa". Online submission.

[17] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. (2019). "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter". arXiv preprint arXiv:1910.01108.

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

[19] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. (2022). "Natural language processing with transformers". " O'Reilly Media, Inc.".

[20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R.mi Louf, Morgan Funtowicz, et al. (2019). "Huggingface's transformers: State-of-the-art natural language processing". arXiv preprint arXiv:1910.03771.