Advances in Science and Technology Research Journal

Advances in Science and Technology Research Journal 2022, 16(3), 207–212
https://doi.org/10.12913/22998624/149941
ISSN 2299–8624, License CC-BY 4.0

Received: 2022.04.30
Accepted: 2022.05.14
Published: 2022.06.01

Ontology Extraction from Software Requirements Using Named-Entity Recognition

Jerzy Kocerka[1,2*], Michał Krześlak[2], Adam Gałuszka[1]

[1] Department of Automatic Control and Robotics, Silesian University of Technology, ul. Akademicka 2A, 44-100 Gliwice, Poland

[2] Tritem Sp. z o.o., Ligocka 103/7, 40-568 Katowice, Poland

* Corresponding author's e-mail: jerzy.kocerka@polsl.pl

ABSTRACT

With the software playing a key role in most of the modern, complex systems it is extremely important to create and keep the software requirements precise and non-ambiguous. One of the key elements to achieve such a goal is to define the terms used in a requirement in a precise way. The aim of this study is to verify if the commercially available tools for natural language processing (NLP) can be used to create an automated process to identify whether the term used in a requirement is linked with a proper definition. We found out, that with a relatively small effort it is possible to create a model that detects the domain specific terms in the software requirements with a precision of 87%. Using such model it is possible to determine if the term is followed by a link to a definition.

Keywords: requirements engineering; ontology extraction; named-entity recognition.

INTRODUCTION

Modern railway vehicles are created with an increasing emphasis on energy efficiency, both due to requirements from the operators regarding the operating costs as well as regulatory requirements caused by environmental issues. The EU Directive on Energy Efficiency (EFD) forces the EU Member States to provide low-energy means of transport [1]. One way to ensure such a goal is to support efficient rail transport, both with traditional electric or hydrogen power supply.

A wide range of energy-saving technologies are used to improve energy consumption, but control software plays a key role in many of them. One of the important parts of quality assurance and risk management, in this case, is software testing and the quality of the requirements. Due to the increasing complexity, the number of functions (and requirements) that require verification in the testing process can be as high as several thousand. Since testing can be one of the most time-consuming parts of the development process (taking 40–70% of the total effort [2]), the process must be as efficient as possible. An important part of this effort is the creation of test cases that require highly skilled engineers who are familiar with the testing process, test environment, and tested domain to be able to analyse and understand the requirements for the system under test. [3]

Most of the software requirements (79 %) are written in the common natural language, such as English, with only 21% using some kind of formalism [4]. Despite many advantages, writing requirements in a common language generates many challenges. The requirement should be precise, unambiguous and complete [5]. Those characteristics are not always ensured when writing in natural languages. Due to this, preparing test cases to verify if the tested software has been properly developed according to requirements, requires high skills, deep analysis and discussions between system engineers, software engineers and test engineers.

The basic condition for a requirement to be precise is the exact definition of the terms used in it. This applies in particular to domains in which specific, specialized terms are used. The railway industry is one example of such an environment. To ensure that terms are unambiguous in many modern application lifecycle management (ALM) tools, it is possible to combine text in requirements with other requirements or descriptions.

In this paper, we propose an automatic process to identify the keywords(specific terms) in software requirements written in natural language to verify if the term is followed by the link to another requirement with a precise definition of the element.

MATERIALS AND METHODS

One of the aims of this study was to verify if the industrial, open-source solutions for processing natural language can be used to identify specific terms in software requirements. We decided to use spaCy [6] – a free and open-source library for advanced Natural Language Processing (NLP) in Python. To identify the specific terms (train elements) in the requirements we used Named-entity recognition – a process that assigns labels to contiguous spans of tokens using a statistical entity recognition system. A named entity is a "real-world object" that's assigned a name – in our case – a train element. SpaCy can recognize various types of named entities in a document, by asking the model for a prediction.

The library has several build-in models to predict the most common named entities like locations, organizations or people, but to identify different kinds of entities we had to teach our model. We've created a set of more than 300 000 paragraphs extracted from project documents. The whole data set was created completely automatically, without any manual intervention, based on Microsoft Word files from the project documentation. The documents were taken from several projects closely related to the analysed domain. Each paragraph, together with the origin meta-data, was treated as a separate document. In total, this gives more than 5 million words.

The set was extended with additional texts from the English Wikipedia. 9219 articles were extracted from Wikipedia, by traversing the category "Rail Transport" [7]. We've selected articles about rail transport in general (e.g. "Rail transport", "Glossary of rail transport terms", "Rolling stock") or about railway vehicles. The articles about the rail infrastructure, rail-related companies or peoples were skipped, as the corpora used in those articles weren't useful for analysing the requirements.

Using both sources, we created a data-set containing over 11 million words related to railway technology, using the vocabulary used in the analysed requirements.

The data was pre-processed to create "senses" [8] and based on such data we trained the vector representation of senses that occur more than 20 times in our corpora. We used fasttext [9] due to its approach, based on the skip-gram model, where each word is represented as a bag of character n-grams. Such an approach should perform better on the corpus with many rare words. As the training data-set was relatively small, we decided to use 100-dimensional vectors. The vector representation was used to create a list of phrases describing train elements which were a basis for our learning process. The list was created by feeding 10 different train elements as a seed and evaluating the most similar phrases. Using this technique, with little effort, we managed to create a list of over 300 train elements.

The annotations for model training were done using Explosion Prodigy tool [10]. In this process, each term was marked (beginning, end) and labelled. Using the tool and the vector-based list described above, we annotated 250 software requirements, each with 0 to 12 different entities. In total, we marked 980 different terms. The use of an items list for pre-selection significantly speed up the process and allowed the annotators to focus on the context of the requirement and on capturing any missing objects. The annotated data were used for model training using spaCy [6]. spaCy uses its own tokenizer to create a tokenized "Doc" out of the raw text and a four-step process [11], shown in Figure 1, to identify entities – non-overlapping, labelled spans of tokens.

The process starts with embedding words into word vectors using Bloom embeddings [12]. Next, the word vectors are converted into a sequence based on their order in the document. Such sequence is an input to the 4-layer residual convolutional neural network (CNN) generating a sequence matrix, where the word meaning is combined with the meaning of its neighbours. The next step in the process (attend) is to reduce the matrix into a single vector and use a feed-forward

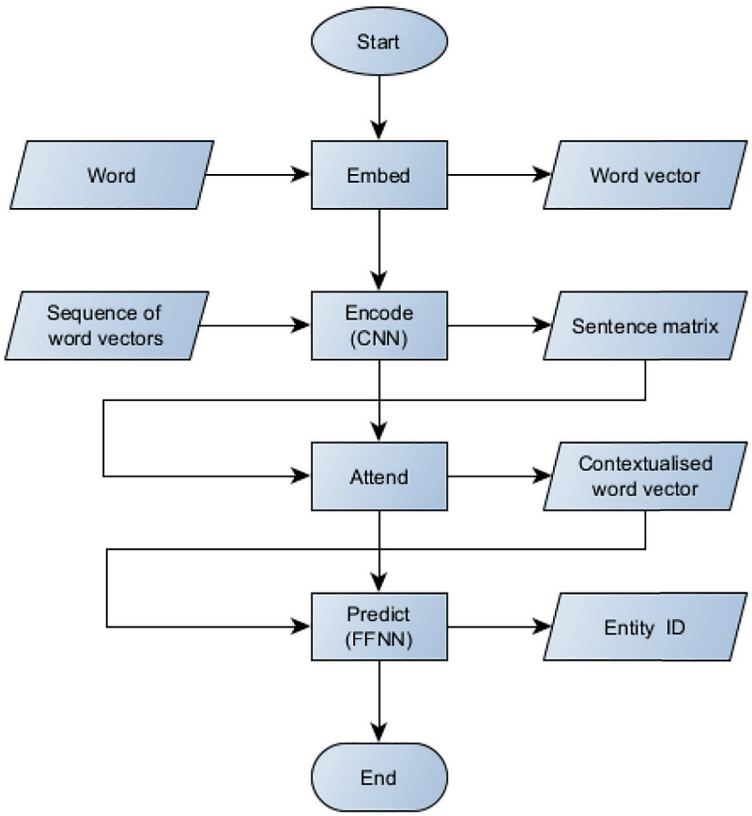

Figure 1. spaCy named entity recognition process

neural network to predict the action related to the word. The possible actions are: beginning of the named entity, inside the named entity, last word of the named entity, outside of the named entity, single word named entity.

As a basis for training, we used a large spaCy model (en_core_web_lg) with embedded vectors (685k unique vectors with 300 dimensions) trained on several, publicly available datasets. As the final step of our analysis, we have created a list of obvious terms that do not require references to their definitions. Then we counted the remaining expressions recognized by the model and compared their number to the number of references in the requirement.

RESULTS

Vector representation

We evaluated the vector representation using an informal qualitative review [13]. As we trained the model to calculate vectors not only for words but also for senses we were able to limit our review to specific parts of speech (nouns, proper nouns) and named entities. We focus on the terms that are specific to railway technology. Table 1 shows a few examples of the most similar terms using the trained vector representation and default, generic spaCy model. As can be seen in Table 1, the vector representation of terms closely related to the railway domain (e.g. "pantograph") indicates railway terms also using the general model. The model learned by us, however, deals much better with terms that also have a general meaning (e.g. "effort").

Using "senses" allowed us to find a vector representation of the items that were already recognized by the pre-processing step as named entities. This approach was very valuable as most of the expressions for elements of a train consist of many words (e.g. brake pipe, pneumatic brake, vehicle control unit).

The evaluation also shows, that with the model trained on our data it's difficult to distinguish between common abbreviations such as UDP, FTP or UTF8 and the acronyms used as names for the train elements such as DCU (Door Control Unit) or ETCS (European Train Control System). As a final evaluation, we compared the output from the trained model to the output of the spaCy *en_core_web_lg* model. We found that we preferred the trained model output in 72% of cases.

Table 1. Similar terms for some railway-related termsDuring the review, we evaluated the 10 most similar items for the selected term. We observed that the first 3 to 5 matches were very similar to the query (with a similarity score usually above 0.5), which was not always true for the rest of the items. This behaviour is probably due to the limited amount of data on which the model was learned. General-purpose vector representations created from publicly available data obtained from the Internet are often learned with several billion words [14].

Word	Trained Vector representation	en_core_web_lg
Brake	Applied, emergency, braking	Brakes, wheel, calliper
Pantograph	Pantographs, raise, raised	Pantographs, catenary, treadle
Effort	Tractive, braking, efforts	Efforts, attempt, helped
Brake pipe	Brake cylinder, pressure, emergency brake module	No vector representation

Named entity recognition

To evaluate whether we generated a training set sufficient to teach the model we've started a learning process for different sizes of batches. In each experiment, 20% of the data was used as an evaluation example. Figure 2 shows the model score depending on the size of the training dataset.

We used the F-measure [15] [16] as a way to check the model accuracy. In general, the F measure is defined as:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \ (0 \leq \beta < +\infty) \qquad (1)$$

where: P is a model precision,
R is a model recall and
β is a parameter that controls the balance between precision and recall.

To evaluate the accuracy of the model prediction we used β = 1, and define our model score as:

$$F_1 = \frac{2PR}{P + R} \qquad (2)$$

By using the spaCy model with embedded vectors we were able to further increase the overall score of the model. In the final experiment, the model achieved an F_1 score of 76.50 % with a precision of 82.35% and a recall of 71.43%.

As the main purpose of our research was to identify objects in requirements to validate, whether they are properly linked with their definitions, we were more interested in high precision as any false positive result may lead to a potential indication of an error where the error was not present. If the term is not recognized by the system (false negative) it will not have a significant influence on the requirement analysis. We conducted a manual analysis of the model output for the evaluation data set and found out that some of the errors were irrelevant considering the purpose. Table 2 shows examples of errors found in model predictions.

More than 20% of the model predictions considered incorrect based on the evaluation data are category 1 errors (incorrect span). Considering the purpose, such errors should be treated as a proper prediction. Additionally, we found three cases in which the model recognized an element that was not marked by the annotator, but after

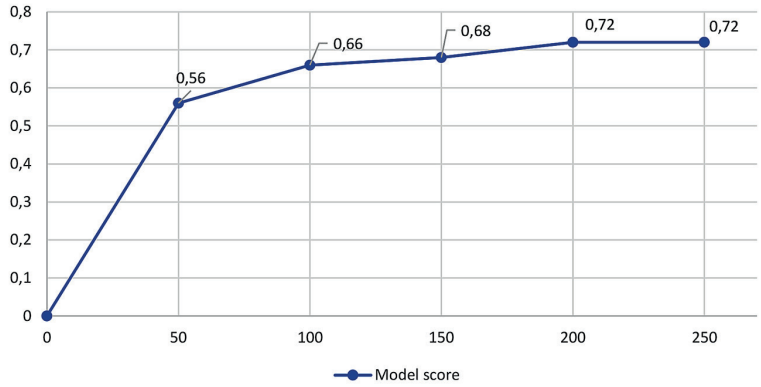

Figure 2. Model F_1 score as a function of training dataset sizeAs visible on the graph, further extension of the training dataset does not improve the model score. Considering that each requirement from the dataset consists of many "terms" the overall number of almost 1000 different examples seems to be enough to teach the model.

Table 2. False positive recognitions

No.	Error type	Example in the evaluation data set	Model output
1.	Incorrect span	In an active cab by the relay k21	In an active cab by the relay k21
2.	Wrong context	Parking brake apply command	Parking brake apply command
3.	Element not part of a train	Brake force applied to rail	Brake force applied to rail
4.	Wrong term	The traction cut-off is not required	The traction cut-off is not required
5.	Missing annotation	For any axle of the bogie	For any axle of the bogie

the revision of the requirement, it was found to be a valid element of the train. Taking this into account, the overall precision of the model precondition was above 87%.

The last step in our analysis was to create a list of all elements detected by the model with the number of their occurrences. On its basis, we manually selected elements that we considered obvious and do not require explanation, e.g. cabin, TCMS etc. Then we counted the occurrences of non-obvious elements in the requirement and compared them with the number of references in the text.

The result shows 3 groups of requirements:

- Very good – number of references close to, or above the number of found elements
- Good – number of references between 30–70% of the found elements
- Insufficient – number of references below 30% of the found elements

In some cases, the score defined above was not applicable, due to none or a very limited number of train elements found by the model in requirement.

CONCLUSIONS

The research has shown that by using publicly available, production-ready tools for natural language processing such as spaCy it is possible to create the model recognising train elements in software requirements written in natural language. The precision of the prediction (above 87%) was high enough to use such a tool non only in research but also in a production environment. The process described in the article requires relatively low effort, as most of the steps are done automatically (generating word vectors, model training) or semi-automatically (annotation with the pre-defined list of items). It can be applied to many different domains, especially if they use their own, specific domain language. The model created with such a process can be used not only for measuring the quality of requirements but also for other tasks (e.g. creating a project glossary).

Acknowledgements

This work has been supported by Department of Automatic Control and Robotics funds for science and development in the year 2022. The calculations were performed with the use of the IT infrastructure of GeCONiI Upper Silesian Centre for Computational Science and Engineering (NCBiR grant no POIG.02.03.01–24–099/13).

Research co-founded by Ministy of Science and Higher Education, agreement: 10/DW/2017.

REFERENCES

1. The European Parliament and The Council of the European Union. Directive (EU) 2018/2002 of the amending Directive 2012/27/EU on energy efficiency, 2018.
2. Kosindrdec N., Daengdej J. A Test Case Generation Process and Technique. Journal of Software Engineering. 2010; 4: 265–287.
3. Kocerka J., Krzeslak M., Galuszka A. Analysing Quality of Textual Requirements Using Natural Language Processing: A Literature Review. In: 2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR) 2018.
4. Mich L., Franch M., Novi Inverardi P. Market research for requirements analysis using linguistic tools. Requirements Engineering. 2004; 9: 40–56.
5. International Council on Systems Engineering. Guide for Writing Requirements, 2019.
6. Explosion. Spacy – Industrial-Strength Natural Language Processing. [Online; accessed 25 November 2021]. Available: https://spacy.io/.
7. Wikipedia. Category:Rail transport – Wikipedia, The Free Encyclopedia. [Online; accessed 12 December 2021]. Available: https://en.wikipedia.org/wiki/Category:Rail_transport.

8. Trask A., Michalak P., Liu J. sense2vec – A Fast and Accurate Method for Word Sense Disambiguation In Neural Word Embeddings. CoRR 2015; abs/1511.06388.
9. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching Word Vectors with Subword Information. CoRR 2016. abs/1607.04606.
10. Explosion. Prodigy – An annotation tool for AI, Machine Learning and NLP. [Online; accessed 25 November 2021]. Available: https://prodi.gy/.
11. Lample G., Ballesteros M., Subramanian S., Kawakami K., Dyer C. Neural Architectures for Named Entity Recognition. CoRR 2016; abs/1603.01360.
12. Serrà J., Karatzoglou A. Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks. CoRR 2017; abs/1706.03993.
13. Denzin N.K., Lincoln Y.S. The SAGE Handbook of Qualitative Research. 5 ed., Sage Publications Ltd., 2017.
14. Pennington J., Socher R., Manning C.D. GloVe: Global Vectors for Word Representation. In: Empirical Methods in Natural Language Processing (EMNLP) 2014.
15. Chinchor N. MUC-4 evaluation metrics. In: Proceedings of the 4th conference on Message understanding – MUC4 1992.
16. Sasaki Y. The truth of the F-measure. 2007.