ECS 412: Project Work
Automatic Word Embeddings-Based Glossary Term Extraction from Large-Sized Software Requirements

Naimish Sharma, 18151

March 19, 2022

1 Introduction

Requirements are the basis for every project, defining what the stakeholders in a potential new system need from it, and also what the system must do in order to satisfy that needs [[1],[2]]. Therefore it is important to have better quality requirements to improve the software product. Quality of requirements could be poor because of different interpretations of technical terms in the requirements by different stakeholders involved in the software development process. Therefore it is important that all stakeholders have similar interpretation of the technical terms used so that overall quality of software gets increased. To cater this, technical(specialized) terms should be defined in glossary. Requirements Glossary can be useful in text summarization and term-based indexing. As manual extraction of glossary terms would be time consuming and expensive task therefore automatic technique is needed.

2 Motivation

In this project I implemented the approach used in paper [3] to extract glossary terms from large-sized software requirements automatically using word embeddings. Candidate glossary terms are extracted using text chunking and then word embedding based semantic filter is used to reduce the number of glossary terms such that only domain specific glossary terms remain in our final list of glossary terms. Domain specific corpus is created by crawling 1576 wikipedia pages related to "Home Automation" category. This technique of glossary term extraction is applied to CrowdRE dataset with 2966 crowd-generated requirements for smart home applications. To evaluate the quality of our extracted glossary terms the ground truth data is manually created from CrowdRE dataset and used for computing precision and recall.

3 Approach

In section Data Gathering, how data is collected is discussed. In section Data Preprocessing, technique used to convert raw text into useful and understable format is discussed. In section Candidate Glossary Term extraction, how using text chunking noun phrases(NPs) are extracted is discussed and In section Semantic Filtering of Candidate Glossary Terms, how word embedding based semantic filter is created is discussed.

3.1 Data Gathering

Let C_{CRE} be the corpus obtained after joining feature and benefit attribute by inserting comma(,) in between and full stop(.) at end of benefit attribute to obtain single textual requirement for each user story in CrowdRE[1] dataset.

[1] https://crowdre.github.io/murukannaiah-smarthome-requirements-dataset/

Let C_{HA} be the corpus obtained after crawling the web pages from "Wikipedia home automation" (HA) category. For web crawling the maximum depth used for subcategory traversal is 2. At first, using PetScan [2] all the page titles upto maximum subcategory 2 are downloaded in csv file and then using Beautiful Soup[3] web scrapping is performed using the page titles i.e. (URL of each page can be obtained by adding "https://en.wikipedia.org/wiki/" and page title from csv file).

3.2 Data PreProcessing

This involves transforming raw natural language text into an understandable format. All the steps of data preprocessing1 have been implemented using the Natural Language Toolkit (NLTK) [4]in Python. Strings such as "[1]", digits, extra white spaces and alpha numeric characters are removed using regular expressions [5]. Textual data (sentences) of each corpus are broken into tokens of words (tokenization). Next, all tokens are converted to lower case, lemmatized(to remove inflection forms) and then stopwords[6] are removed. Let C'_{CRE} and C'_{HA} be the corpus obtained after applying above text preprocessing techniques on C_{CRE} and C_{HA}.

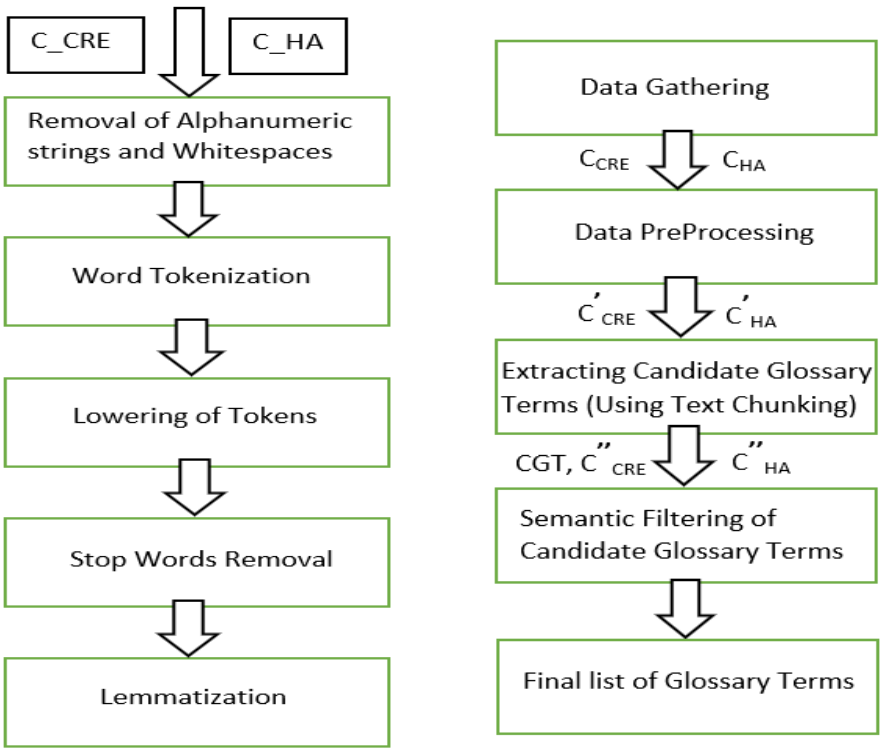

Figure 1: Data PreProcessing Pipeline and Semantic Filtering Approach for extracting Glossary Terms

3.3 Candidate Glossary Term Extraction

Using NLTK ($pos - tag$) Tagger part of speech of each token is obtained. Then grammer is defined as $\{< DT >? < JJ >^* < NN >\}$ i.e. one or zero determiner followed by zero or more adjectives and a noun. Text chunking is performed on both corpus based on this grammer to obtain Noun Phrases(NP). GT is the set of Noun Phrases(NPs) obtained from C'_{CRE} and TW is the set of Noun Phrases(NPs) obtained from C'_{HA}. Let CGT = GT $\cap$ TW.

[2] https://petscan.wmflabs.org/
[3] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[4] https://www.nltk.org/
[5] https://docs.python.org/3/howto/regex.html
[6] https://www.ranks.nl/stopwords

3.4 Semantic Filtering of Candidate Glossary Terms

CGT contains the NPs on which semantic filtering has to be applied. C'_{CRE} is transformed to C''_{CRE} by replacing each occurence of NP that appeared in CGT with modified version of NP. Modified version of noun phrase is obtained by replacing space(' ') with underscore('_') and adding dollar('$') before and after the token. Also, C'_{HA} is transformed to C''_{HA} by replacing space(' ') with underscore('_') in noun phrases with more than one word.
Both Skip-gram and Continuous Bag of Words (CBOW) Word2Vec models are used to to produce word embeddings and for computing semantic similarity scores of NPs present in CGT. Similarity Score will help in determining if a word is used in similar context in C''_{CRE} and C''_{HA} corpus or not. Corpus used to learn word embeddings is $C''_{CRE} \cup C''_{HA}$. Vector_size used is 100, window size used is 10, and the minimum count (c = 1) for all the experiments. Using cosine similarity, sematic relationship of two words is computed in the vector space i.e. how close or far are they in vector space. Glossary terms with similarity more than 0.85 are only included in final list of glossary terms as I wanted to include terms in glossary that are highly domain specific.

4 Analysis of Results and Discussion

Descriptive Statistics of the data used is discussed here, Table1. R means requirements in CrowdRE dataset and P means wikipedia pages. Similarity Score using (word embeddings) by SkipGram and CBOW vs Number of NPs(in CGT) greater than similarity score is discussed in Table2. Number of NPs in CGT i.e. common in both C''_{CRE} and C''_{HA} is 1800. We can observe that using only text chunking there are 1800 glossary terms overall whereas using word embedding based semantic filter (with similarity value greater than 0.85) glossary terms in our final list reduces to 262 and 243 (including some words not related to home automation) using SkipGram and CBOW word embeddings respectively. Ground Truth is generated manually over first 100 requirements and then Precision and Recall are calculated by extracting candidate glossary terms by text chunking(NPs). Recall and precision are around 78% and 22% respectively. Precision is low because we used only text chunking to extract candidate glossary terms from subset of requirements. There are many not domain specific words extracted from the requirements while text chunking. All reports and codes are available at repository [7]

Table 1: Descriptive Statistics of the data

Data	No. of documents	Total tokens after preprocessing	Total NPs
C_{CRE}	2966(R)	31850	4992
C_{HA}	1576(P)	699169	78640

Table 2: Similarity Score using (word embeddings) by SkipGram and CBOW vs Number of NPs(in CGT) greater than similarity score

Similarity score	Using SkipGram Word Embeddings	Using CBOW Word Embeddings
0.9	162	102
0.85	262	243
0.8	352	384
0.7	507	656
0.6	723	904
0.5	983	1086

[7] https://github.com/naimish3199/Automatic-Word-Embeddings-Based-Glossary-Term-Extraction-from-Large-Sized-Software-Requirements

$$Precision = \frac{TP}{TP + FP} = \frac{|\{relevant\,documents\} \cap \{selected\,documents\}|}{|\{retrieved\,documents\}|} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} = \frac{|\{relevant\,documents\} \cap \{selected\,documents\}|}{|\{relevant\,documents\}|} \quad (2)$$

5 Conclusion and Future Work

Word Embedding based semantic filter helped in reducing number of candidate glossary terms from 4992 to around 262 from 2966 requirements. Some examples of extracted glossary terms and their similarity scores are given here Table3. One important point is that we even counted terms as candidate glossary terms which are rare and appeared maybe once(Ex - smart tv, smart fridge, room thermostat, iphone etc) in our corpus so we can include some frequency criteria also in our approach. We can include top n terms in our final glossary list based on the threshold value of similarity score. Limitations of this approach is that it requires large amount of domain specific corpus to design semantic filters. So, many new emerging domains have sortage of domain specific corpus on web therefore this approach will not work.

Further we can use BERT, LSTM, fastext to obtain word embeddings and ensure that only domain specific terms are there in final list of glossary. Also as we only extracted domain specific corpus from wikipedia, more domain specific corpus can be generated using other websites, magazines and articles. Even using OCR(Optical Character Recognition), text can be extracted from scanned hard copies of domain specific documents and can be used in our domain specific corpus. Many pages we extracted from wikipedia were not domain specific but we still included in Home Automation corpus. So, to get rid of this problem we can design some filter to include a wikipedia page in our corpus only if it is domain specific.

Table 3: Examples of final extracted glossary terms and their similarity scores

Glossary Terms	Similarity score
alert security	0.9472
automatic lighting	0.9189
coffeemaker	0.9100
efficient power	0.9363
electric blanket	0.9355
interior temperature	0.9405

References

[1] Elizabeth Hull, Kenneth Jackson, and Jeremy Dick. *Requirements engineering in the solution domain.* Springer, 2005.

[2] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques.* Springer Publishing Company, Incorporated, 2010.

[3] Siba Mishra and Arpit Sharma. Automatic word embeddings-based glossary term extraction from large-sized software requirements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 203–218. Springer, 2020.