From word embeddings to text similarities for improved semantic clustering of functional requirements

Takwa Kochbati, Sébastien Gérard, Shuai Li, Chokri Mraidha
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
{takwa.kochbati, sebastien.gerard, shuai.li, chokri.mraidha}@cea.fr

Abstract— **Requirements engineering starts by requirements elicitation which consists in gathering software requirements from stakeholders. Then, the elicited requirements are usually manually recorded in a requirements specification document. In recent years, modern software projects are becoming more complex than projects of the past due to the increase in the number of requirements and stakeholders involved in a project. Thus, manually managing requirements becomes a tedious, time consuming and error-prone task. One historical strategy to manage this kind of complexity is "divide to conquer", meaning to categorize them into groups in order to breakdown the system into a set of smallest sub-systems at early stages. In this paper, we propose an approach to automatically cluster functional requirements based on their semantic similarity which is the usual strategy used by system architects to define sub-systems candidate to simplification of the original problem. First, we use word2vec, as a predictive word embedding model to compute the word-level similarity. Second, we derive the requirement-level similarity using a scoring function for text similarity. Third, we adopt hierarchical clustering to group the requirements. Experimental results performed on four open-access software projects show that our approach succeeded to improve the results of clusters identification compared with existing studies.**

Keywords-component; software requirement; clustering; word embedding; natural language processing.

I. INTRODUCTION

Requirements elicitation is the first step in developing a software product. In this step, engineers discover and collect requirements from customers and then, they manually record them in a requirements specification document. The gathered requirements describe different aspects of the target software in natural language and they are mainly classified into functional and non-functional requirements [1]. Functional requirements describe the functional behavior and the features of the software system while non-functional requirements define the system attributes such as performance, security, reliability as well as the system operational conditions such as power consumption and environmental conditions.

Requirements elicitation has a significant impact on information systems quality and success, as the errors introduced at the beginning stages of development are the hardest and most expensive to correct [2].

Hence, it is crucial that the requirements set has to be well understood and well managed by engineers [3].

System design constraints evolves more and more requiring to embed more stakeholders in the projects to handle various new concerns - such as security, safety, cost, and sustainability – earlier in the process, at specification time. Consequently, modern software projects are becoming many times larger and hence more complex than in the past. Especially, the exponential growth of the number of requirements raises difficulties in managing manually the requirements and having a clear crystal view of the expectation and scope of the system to be designed [4]. One of the most used and efficient design paradigms to deal with complexity is the well-known "divide-to-conquer" strategy i.e., building smallest pieces to reduce the complexity. Herein lies the importance of an automatic solution to categorize software requirements into a set of groups in order to breakdown the target software system into a set of smallest sub-systems at early stages of the development process.

In this paper, we propose a clustering solution to automatically group functional requirements based on their semantic similarity. We use and analyze the semantic information of the requirements to compute the requirements similarity at two levels: at the word level, but also at the statement level of the requirements (i.e., local versus global semantics of the requirements). In that context, we make the following contributions: 1) we use a neural word embedding model, word2vec, as a predictive model to compute the word-level similarity; 2) then, we derive the requirement-level similarity using a scoring function for text similarity computation; 3) finally, we adopt hierarchical clustering combined with a pre-defined criteria to group the requirements in specific clusters. To evaluate our proposal, we have successfully applied it to four open-access software projects.

The remainder of the paper is structured as follow: section 2 discusses the related works; section 3 describes the proposed approach; section 4 provides the experimental evaluation settings; section 5 provides the results analysis; section 6 raises the limitations and the threats to validity and finally, section 7 concludes the paper.

DOI reference number: 10.18293/SEKE2021-056

II. RELATED WORKS

In recent years, the usage of clustering techniques in the early phases of software engineering has gained a lot of attention.

In [5], the authors developed a tool based on hierarchical clustering of requirements in order to propose a packaging solution for software engineers. They defined a similarity measure that aims to cluster classes with high number communication in the same package. The optimal number of clusters is manually selected by software engineers based on the hierarchical tree generated by the clustering algorithm.

In [6], the authors present an approach based on concepts clustering to visualize requirements at different levels of granularity. They employed *word2vec* as a predictive model to compute similarity between concepts.

The authors in [7] propose an initial clustering of responsibilities from requirements, in order to detect architecture components. The approach is validated using four different clustering algorithms and several validity metrics. The similarity function is computed according to the verb phrase each responsibility contains, and the direct object it is related to.

In [8], the authors present an approach to cluster and sequence user stories in order to assist software engineers in the implementation phase. They employed clustering algorithm and the silhouette score to identify the best clustering solution.

In [9], the authors propose an approach that clusters similar requirements in order to reuse them in software product lines (SPLs). They compared the performance of two clustering algorithms based on a distance measure in order to identify similar requirements.

In [10], the authors demonstrate the use of the HAC algorithm to group functional requirements based on their similarity. Their work aims to breakdown the project into a set of sub-projects at early stages. They use traditional vector space models (VSMs) to vectorize text requirements and use the *cosine similarity* to measure the semantic similarity between requirements.

All these techniques inspire our work. However, some of these approaches suffer from a lack of automation as for example when defining the optimal number of clusters [5], others rely on the similarity between words or concepts in each requirement [6], [7]. Moreover, many works rely on traditional distributional semantic models (DSMs), for instance Vector Space Model (VSM) [8], [10] and Latent Semantic Analysis (LSA) [9] to calculate the similarity. The main limitation of these techniques is that they are considered as “count” models as they rely on counting the co-occurrences among words by operating on co-occurrence matrices. Thus, sentences with similar context but different term vocabulary will not be considered as similar. Consequently, traditional DSMs usually achieve worse results than neural word embedding models, which can be seen as predictive models [11].

The main novelty of our proposal is that we benefit from using the neural word embedding model *word2vec* as predictive model, to compute word level similarity and then, derive the requirement level similarity using a scoring formula for text similarity.

III. THE PROPOSED APPROACH

In this section, we explain how our approach processes in order to generate automatically the clusters from natural language requirements as illustrated in the process shown in Figure 1.

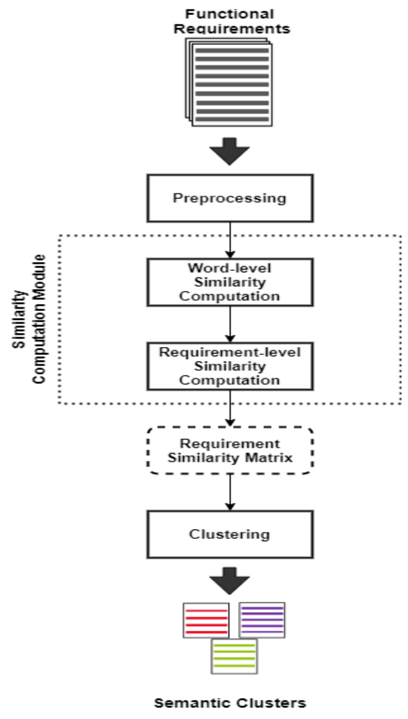
Functional Requirements
Preprocessing
Similarity Computation Module
Word-level Similarity Computation
Requirement-level Similarity Computation
Requirement Similarity Matrix
Clustering
Semantic Clusters

Figure 1. Overview of the approach

In what follows, we detail the particular techniques used in each step.

A. *Preprocessing*

Preprocessing is the first step of the approach in which, the input functional requirements expressed in natural language are normalized through four steps: (i) tokenization, i.e., the decomposition of a sentence into a set of individual words; (ii) stop-words removal, i.e., the elimination of common English words; (iii) punctuation removal; (iv) stemming, i.e., the transformation of each word to its root (e.g: "adding" becomes "add").

B. *Semantic similarity computation module*

The preprocessed requirements are then introduced into the semantic similarity computation module. Traditional approaches to compute the similarity between two text segments consist in using lexical matching method, and producing a similarity score based on the number of lexical units that occur in both input segments. However, these lexical similarity methods cannot always identify the semantic similarity of texts as they aim to determine whether the words in two texts have similar spellings [12]. For example, the "*US*" would be closer to the "*UK*" this way, than it would be to the "*States*".

Going beyond these traditional methods, we compute and analyze the semantic information at two levels: locally, for each word contained in a requirement description, but also globally at the statement level.

1) Word-level similarity computation

In order to compute the word-level similarity, one must rewrite the preprocessed requirements from natural language to a machine-readable and analyzable format. Thus, words should be transformed into numerical vectors that work with machine learning algorithms. To this end, we use the *word2vec* model, a two-layer neural network that is used to produce word embeddings (i.e., vectors).

The input of *word2vec* is a text corpus. Given enough text data and contexts, *word2vec* can achieve highly accurate semantics of the words appearing in the corpus and establish a word's association with other words in the semantic space. Moreover, word embedding models have shown to outperform traditional DSMs which are considered as "count" models as they count co-occurrences among words by operating on co-occurrence matrices [11].

Since a word embedding model is supposed to be of high quality when trained with large corpus, we use the pretrained *word2vec* model on 100 million words of *Google News* dataset (https://code.google.com/archive/p/word2vec/). However, even if the used corpus is large (e.g., *Google News*), some domain-specific words founded in the requirement statement may be unknown in the corpus. In this case, as suggested in [13], we assign a random vector to the missing word. Then, we compute the semantic similarity between each pair of the obtained word vectors belonging to two different requirement statements using the *cosine similarity* measure. The *cosine similarity* principle consists in computing the cosine of the angle between two words vectors. Thus, the *cosine similarity* of two similar words vectors is close to 1, and close to 0 otherwise.

2) Requirement-level similarity computation

After obtaining the word-level similarity, we extend it at the global statement-level. Some approaches capture the meaning of longer pieces of text by taking the means of the individual term vectors [14], [15]. However, means or sums are rather poor ways of describing the distribution of word embeddings across a semantic space. It would be desirable to capture more properties of the two texts, especially with respect to the semantics of words that do or do not match.

We overcome the above-mentioned limitations by deriving the statement-level similarity from the word-level similarity based on two characteristics: the distribution of words in each requirement statement; and the specificity of each word in the requirements document. To do that, we got inspiration from the work of Mihalcea et al. [12], to derive the statement-level semantic similarity from the word-level semantic similarity. We used hence the Mihalcea's scoring function for text similarity computation to compute the similarity of each pair of requirement statement (see Equation 2).

First, we identify for each word w_1 in the text requirement R_1, the word w_2 in the text requirement R_2 that have the highest semantic similarity *maxSim*(w_1,R_2) (Equation 1), based on the word-to-word semantic similarity *wordSim*(w_1,w_2) using *word2vec*. Next, the same process is applied to determine the most similar word in R_1 starting with words in R_2.

$$maxSim(w_1, R_2) = \max_{w_2 \in R_2} wordSim(w_1, w_2) \qquad (1)$$

The word similarities are then weighted with the corresponding word specificity using the *Inverse Document Frequency (idf)* weighting technique to capture the specificity of a word. In a nutshell, this technique aims to measure how much a word contributes to the relevance of two texts. The weighted word similarities are then summed up and normalized with the length of each text segment. The resulting similarity scores are combined using a simple average and thus, the semantic similarity of two requirements R_1 and R_2 is computed as follows:

$$sim(R_1, R_2) = \frac{1}{2} \times \left(\frac{\sum_{w \in R_1} maxSim(w, R_2) \times idf(w)}{\sum_{w \in R_1} idf(w)} + \frac{\sum_{w \in R_2} maxSim(w, R_1) \times idf(w)}{\sum_{w \in R_2} idf(w)}\right) \qquad (2)$$

Ultimately, by applying the equation (2), we obtain the final similarity matrix of each pair of requirements.

C. Clustering:

Textual requirements clustering refers to the process of taking a set of requirements and grouping them based on a similarity measure so that, requirements in the same cluster are similar and requirements in different clusters are different. In this context, we adopt the clustering of functional requirements based on their semantic similarity.

Clustering methods can be classified either as hierarchical or partitional [16]. Partitional clustering algorithms such as *k-means*, require the number of clusters. Thus, they rely heavily on the analyst's knowledge, as they require the identification of the number of clusters to be generated in advance. In order to reduce the manual intervention, we employ the Hierarchical Agglomerative Clustering algorithm (HAC) [17] as it does not require us to pre-specify the number of clusters in advance. Hence, we utilize the similarity values for each pair of requirements as clustering criterion, taking the semantic similarity matrix of the functional requirements as input for HAC. The HAC algorithm works in a bottom-up manner, each requirement statement is initially considered as a single-element cluster (leaf). At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (node). This procedure is iterated until all requirements are member of just one single big cluster, resulting in a hierarchical clustering tree.

However, identifying the optimal number of clusters is not a trivial task. It might be subjective as it can heavily rely on the analyst's knowledge. In order to automate this task, we implement an operation that identifies automatically the best number of clusters using the *Dunn index* [18]. The *Dunn index* is an internal validity index used to evaluate the clustering result when the number of clusters is unknown. Hence, in order to achieve an optimal number of clusters, we calculate the *Dunn index* each time, when varying the number of clusters. A higher *Dunn index* indicates better clustering solution. Consequently, to estimate optimal number of clusters that are generated by HAC, we select the number of clusters for which we have a higher *Dunn index.*

IV. EXPERIMENTAL EVALUATION SETTINGS

In order to assess our approach, we report in this section the research questions that were investigated as well as the four case studies we did.

A. *Research questions:*

As our study focuses on the automatic grouping of the functional software requirements into a set of clusters, we investigated the following research questions to evaluate the approach:

- ***RQ1: To what extent is the proposed clustering solution accurate?***

Motivation. For this research questions, we aim at determining the accuracy of the proposed clustering solution in order to assess whether our approach succeeded to identify semantic clusters that reflect the domain functionalities embedded in a given functional requirements document**.**

Approach. To answer this research question, we evaluate the proposed clustering solution using two validation criteria as follows:

- *The correctness of the identified semantic clusters:*

This validation criterion aims at verifying whether the identified semantic clusters are close to the semantic clusters provided in the software requirements specification (SRSs) documents. For this, we rely on two well-known measures in the Information Retrieval (IR) field. These metrics are *precision* and *recall* [19]. The identified clusters are compared with the reference clusters provided in the SRSs documents, which serve as a ground truth for our evaluation.

Let True Positive (TP) elements be the similar requirements correctly assigned to the same cluster, False Positive (FP) elements be dissimilar requirements assigned to the same cluster and False Negative (FN) elements be similar requirements incorrectly assigned to different clusters. The evaluation metrics are defined as follows:

$$\textit{Precision} = TP / (TP + FP)$$

$$\textit{Recall} = TP / (TP + FN)$$

- *The clustering gap (C_GAP):*

For this validation criterion, we aim to verify whether the identified number of clusters is close to the reference number provided in the SRSs document. This is recognized as the clustering gap (*C_Gap*). The *C_Gap* compares the identified number of clusters with the reference number of clusters. Thus, it is defined as follows:

$$C\_Gap = |number_{identified\_clusters} - number_{reference\_clusters}|$$

- ***RQ2: Is the proposed clustering solution practical in realistic settings?***

Motivation. For this research question, we aim to establish whether our approach is scalable. Particularly, the goal is to check how well the clustering solution performs when increasing the number of functional requirements.

Approach. In order to solve this research question, we assess the following validation criterion:

- *The end-to-end execution time of the clustering solution:*

It consists in measuring the impact of the number of software requirements for each case study on the execution time. Hence, this validation criterion aims to check whether the proposed clustering solution runs within reasonable time for larger number of functional requirements in realistic settings.

B. *Case studies:*

We assess the applicability of our approach using the software requirements specification documents of four open-access projects from different domains and with different sizes: the E-Store software consists of online sales, distribution and marketing of electronics [20]. The WASP system is a public, real-world requirements specification of context-aware mobile telecommunication services [21]. The UUIS system - Unified University Inventory System - is used to integrate three faculties' databases providing a web interface that allows user to access and manage the integrated inventory [20]. The MHC-PM system is a Mental Health Care Patient Management System [22].

The table below shows the characteristics of each case study in terms of number of requirements as well as the number of clusters in each SRSs document.

TABLE I. Characteristics of the Case Studies

Case study	Number of requirements	Number of clusters
E-Store system	62	20
WASP system	66	14
UUIS system	25	11
MHC-PM system	19	6

V. RESULTS ANALYSIS

In this section, we evaluate the results of applying our proposal to the four aforementioned case studies through the two previous described RQs.

A. *Answering RQ1: To what extent is the number of identified clusters correct?*

- *The correctness of the identified semantic clusters:*

In Tables II, we present an example of the reference and the identified cluster for the E-Store system. The requirement statement shown in bold in the identified cluster is an irrelevant functional requirement in that cluster.

In order to answer RQ1, we evaluate our clustering results in terms of *precision* and *recall*. We also compare our results to the work in [10]. In fact, the approach used in [10] closely relates to our work as it proposes a method to semantically cluster functional requirements. Thus, we use the work in [10] as a baseline. Table III shows the evaluation results of our approach as well as the baseline [10] in terms of *precision* and *recall*. The best results are in bold.

TABLE II. Example of identified and reference cluster for the E-store system

Identified cluster	Reference cluster
The system shall allow user to create profile and set his credential.	The system shall allow user to create profile and set his credential.
The system shall authenticate user credentials to view the profile.	The system shall authenticate user credentials to view the profile.
The system shall allow user to update the profile information.	The system shall allow user to update the profile information.
The system shall allow user to register for newsletters and surveys in the profile.	

Precision values take a high-range (0.74 – 0.87) and *recall* values take a reasonable range (0.63 – 0.75) across different case studies. In most of these case studies, we achieved better *precision* and *recall* values compared with the baseline [10]. For example, for the MHC-PM case study our approach achieves better *precision* and *recall* values by 6 and 16 percentage points respectively. Thus, the evaluation shows clustering results with relatively high quality with better *precision* and *recall* values in most case studies compared with the baseline [10].

TABLE III. Precision, Recall and C_Gap Values for each Case Study

		E-store	WASP	UUIS	MHC-PM
Our approach	**Precision**	**0.83**	**0.87**	**0.74**	**0.84**
	Recall	**0.68**	**0.63**	**0.75**	**0.73**
	C_Gap	**0**	**2**	**1**	1
The baseline	**Precision**	0.80	0.83	0.72	0.78
	Recall	0.61	0.54	0.60	0.57
	C_Gap	1	4	2	**0**

At the light of these results, the answer to the first research question (RQ1) is that our clustering solution succeeded to achieve relatively accurate results that can be applicable.

- *The clustering gap (C_GAP):*

TABLE III shows, for each case study the clustering gap between the identified and the reference clusters. By comparing these results with TABLE I, we note that the identified number of clusters is the same or very close to the reference number of clusters. Moreover, by comparing with the baseline [10], our method succeeded to identify a number of clusters that is closer to the reference number of clusters for the three case studies: E-store system, WASP system and UUIS system.

Therefore, the answer to this research question is that the identified number of clusters is very similar to their corresponding reference number of clusters. In summary, we conclude that the number of the identified cluster is accurate and achieves better results than the baseline [10] in most case studies.

B. *Answering RQ2: Is the proposed clustering solution practical in realistic settings?*

In order to answer this research question, we measure the execution time of our approach for the four case studies on a laptop with a *2.10 Ghz Intel (R) Core (TM) i7-4600U CPU* and a *8GB* of memory. In Table IV, we measure the impact of the number of requirements on the end-to-end execution time in order to assess the applicability of our solution.

TABLE IV . Execution Time by Number of Requirements

	MHC-PM	UUIS	E-Store	WASP
Number of requirements	19	25	62	66
Execution time in seconds	20	24	41	52

Table IV shows that our approach runs in few seconds for the four case studies. Moreover, Figure 2 shows a linear growth trend for the impact of the number of requirements on the execution time. Given such linear relation and the fact that the end-to-end execution time takes few seconds, the answer to RQ2 is that our approach runs in reasonable time.

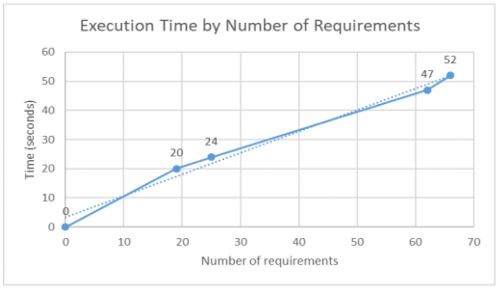

Figure 2. Execution Time by Number of Requirements Graph

In summary, we anticipate that our clustering solution should be practical for much requirements documents.

VI. THREATS TO VALIDITY

In this section, we discuss the limitations of our proposal in terms of internal threats, construct threats and conclusion threats. These threats are as follows:

Internal validity. With regard to computing word similarity, some domain-specific words do not occur in the corpus used to train the word vector space, which might slightly affect the efficiency of word similarity computation. To mitigate this limitation, we map such words to a random vector.

External validity. Our approach is capable of generating clusters from short text requirements. However, if a requirement describing a functionality is of too many sentences, our approach maybe cannot provide an accurate result. Hence, in this case, some manual semantic analyses may still be needed to overcome this limitation.

Conclusion validity. We evaluate the applicability of our approach on four open-access projects. Although the evaluation results are promising, the results from just four domains may be not enough to support the conclusion. Thus, we need to evaluate the approach on larger number of case studies for a better evaluation.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach to automatically group functional requirements into semantic clusters in order to breakdown automatically a system into sub-systems at early stages, providing to the system architect a first high-level architecture description of her/his system.

The core of the approach is a clustering solution that is based on the semantic similarity of the natural language requirements. In order to improve the accuracy of the clustering solution, semantic information of both the words and requirements is analyzed and used for compute the similarity. Word-level similarity was firstly computed using *word2vec* as pre-trained predictive model then, it was extended to the requirements-level using the Mihalcea scoring formula for text similarity computation. Then, we employ the HAC algorithm to cluster functional requirements into semantic clusters. Moreover, we propose and automatic identification of the optimal number of clusters in order to reduce the manual intervention.

To assess the applicability of our approach, we conduct four case studies from open-access projects from different domains and evaluate the results in terms of *precision*, *recall* and execution time. Evaluation results reveal that we succeeded to achieve relatively accurate semantic clusters fully automatically within a practical execution time that takes few seconds. Moreover, a comparison with a related work shows that our approach provides relatively better clustering results.

As future work, we will focus on extending our clustering solution to categorize non-functional requirements according to their type. Indeed, most of the work focusing on automating non-functional requirements categorization use supervised learning techniques requiring huge training datasets, which are not always available for all domains. So far, employing clustering to categorize non-functional requirements did not provide sufficient accuracy. Hence, we plan to integrate other techniques taking into account popular key words of each non-functional requirement type to enhance the categorization process.

REFERENCES

[1] R. Pressman, ”Software Engineering: A Practitioner’s Approach ,” 1982.

[2] B. Brügge and A. Dutoit, “ Object-Oriented Software Engineering Using UML, Patterns, and Java ,” 2009.

[3] D. Zowghi and C. Coulin, “Requirements Elicitation: A Survey of Techniques, Approaches, and Tools”, in *Engineering and Managing Software Requirements*, Springer, Berlin, Heidelberg, 2005, p. 19-46.

[4] F. Brooks, “No Silver Bullet Essence and Accidents of Software Engineering”, *Computer*, vol. 20, n^{o} 4, p. 10-19, avr. 1987, doi: 10.1109/MC.1987.1663532.

[5] Y. Amannejad, M. Moshirpour, B. H. Far, and R. Alhajj, “From requirements to software design: An automated solution for packaging software classes”, in Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014), août 2014, p. 36-43.

[6] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, et S. Brinkkemper, « Visualizing User Story Requirements at Multiple Granularity Levels via Semantic Relatedness », in Conceptual Modeling, nov. 2016, p. 463-478.

[7] A. Casamayor, D. Godoy, et M. Campo, “Functional grouping of natural language requirements for assistance in architectural software design”, Knowl. Based Syst. 30, 2012, 78-86.

[8] R. Barbosa, D. Januario, A. E. Silva, R. Moraes, et P. Martins, “An Approach to Clustering and Sequencing of Textual Requirements”, 2015 IEEE International Conference on Dependable Systems and Networks Workshops : 39-44, 2015.

[9] H. Jalab and Z. M. Kasirun, “Towards Requirements Reuse: Identifying Similar Requirements with Latent Semantic Analysis and Clustering Algorithms”, 2014.

[10] H. E. Salman, M. Hammad, A.-D. Seriai, and A. Al-Sbou, “Semantic Clustering of Functional Requirements Using Agglomerative Hierarchical Clustering”, Inf. 9 (2018): 222., 2018.

[11] M. Baroni, G. Dinu, and G. Kruszewski, “ Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors”, ACL, 2014.

[12] R. Mihalcea, C. Corley, and C. Strapparava, “ Corpus-based and Knowledge-based Measures of Text Semantic Similarity “, AAAI, 2006.

[13] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” EMNLP, 2014.

[14] B. Hu, Z. Lu, H. Li, and Q. Chen, “Convolutional Neural Network Architectures for Matching Natural Language Sentences,” NIPS, 2014.

[15] R. Socher, D. Chen, C. D. Manning, and A. Ng, “Reasoning With Neural Tensor Networks for Knowledge Base Completion,” NIPS, 2013.

[16] M. Allahyari et al., “A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques,” ArXiv abs/1707.02919, 2017.

[17] M. L. Zepeda-Mendoza and O. Resendis-Antonio, “Hierarchical Agglomerative Clustering,” in Encyclopedia of Systems Biology, Springer, New York, NY, 2013, pp. 886–887.

[18] J. C. Dunn, “Well-Separated Clusters and Optimal Fuzzy Partitions,” 1974.

[19] C. D. Manning, P. Raghavan, and H. Schütze, “Introduction to information retrieval,” 2005.

[20] National Research Council of Italy. Natural Language Requirements Dataset. Available online: http://fmt.isti.cnr.it/nlreqdataset/ (accessed Feb. 22, 2021).

[21] T. Menzies, R. Krishna, Pryor “PROMISE Software Engineering Repository.” http://promise.site.uottawa.ca/SERepository/ (accessed Feb. 22, 2021).

[22] Mental Health Care Patient Management System. Available online: https://bscs143.files.wordpress.com/2015/11/requirement-mhc-pms.docx (accessed Feb. 22, 2021).