ICIC Express Letters
Volume **16**, Number **9**, September **2022**

 ISSN 1881-803X
pp. **913–921**

COMBINETF FOR REQUIREMENTS DATA SIMILARITY DETECTION ON AREM

Rosa Delima[1,2], Retantyo Wardoyo[1,*] and Khabib Mustofa[1]

[1]Department of Computer Science and Electronics
Universitas Gadjah Mada
Sekip Utara Bulaksumur, Yogyakarta 55281, Indonesia
khabib@ugm.ac.id
*Corresponding author: rw@ugm.ac.id

[2]Department of Informatics
Universitas Kristen Duta Wacana
Dr. Wahidin Sudirohusodo Street 5-25, Yogyakarta 55224, Indonesia
rosadelima@staff.ukdw.ac.id

Received September 2021; accepted December 2021

Abstract. *The Automatic Requirements Engineering Model (AREM) is a model that can automate the requirements engineering process. This model accepts input in the form of requirements data from several stakeholders. The similarity of the description of the requirements of one stakeholder with other stakeholders is very likely to occur. Therefore, the collected requirements data are to be processed and tested for similarity so that there is no duplication of requirements in system modeling. In this study, the CombineTF method was developed to check the similarity of the data requirements. CombineTF is a hybrid method that combines a term-based approach with Term Frequency (TF) and character-based similarity. In this research, CombineTF is integrated with the Jaro-Winkler algorithm and Levenshtein distance as a character-based similarity. The experimental results show that CombineTF has a good performance for measuring the similarity of requirements documents with a threshold of more than 0.5.*

Keywords: CombineTF, Jaro-Winkler, Levenshtein distance, Requirements engineering, Term frequency

1. **Introduction.** Requirements Engineering (RE) is a crucial stage for modeling requirements in the software engineering process. This process consists of several stages, including elicitation, analysis, specification, and validation. The Automatic Requirements Engineering Model (AREM) is an automatic requirement engineering model developed by using an oriented goal approach. This model was developed to be able to automate four processes in RE, i.e., elicitation, analysis, specification, and validation.

In the requirements elicitation stage, the requirements of stakeholders are collected. Stakeholders may consist of various elements, such as the organization's leaders, managers, and end-users. The requirements of each stakeholder will be stored in the requirements data. All of the requirements data from all stakeholders will be incorporated into an input for AREM to do a requirements analysis.

Requirements analysis is done by extracting the requirements data collected at the elicitation stage. In the process of requirements extraction, it will be found that there are similarities between the requirements of one stakeholder and other stakeholders. Therefore, it is important to detect similarities in the description of requirements between stakeholders so that there is no duplication of system requirements that will affect the quality of the requirements specifications produced by AREM.

DOI: 10.24507/icicel.16.09.913

To perform the detection of the similarity of requirements, in AREM, we developed a method that combines the calculation of Term Frequency (TF) by the method of similarity-based characters. This method is named CombineTF. The TF approach is a simple approach to detecting the occurrence of the same term in two documents [1]. The term is a representation of a word. In the requirements data, the use of the same term by several stakeholders is very often encountered; therefore, the detection of the same term will be very influential in testing document similarity. In this study, TF was combined with two character-based methods, i.e., Jaro-Winkler and Levenshtein distance.

Jaro-Winkler (JW) method is a development of the Jaro distance with the addition of the prefix string matching calculation [2,3]. This method has good accuracy for detecting similarities in two strings [4,5]. Meanwhile, Levenshtein distance is a method of measuring the similarity of two strings using a matrix. This method calculates the minimum number of operations required to equalize the two strings [6]. This method has been applied extensively and developed with various methods and other approaches. The CombineTF, which is integrated with JW or Levenshtein, contributes to the development of a hybrid method that integrates character-based and token-based similarity. This method has the advantage of detecting the similarity of two texts that have many similar tokens. This method will be able to improve AREM's ability to detect similar requirements from stakeholders so as to avoid duplication of requirements and improve the quality of requirements modeling.

This paper contains the development of the CombineTF method to detect the similarity of requirements documents in AREM. This paper is organized into five parts, that is, introduction, related work, the development of the CombineTF method, experiments, and results, and the final part is closed with a conclusion.

2. **Related Work.** The requirement data at AREM is in the form of text. Text similarity detection is needed to determine the level of similarity in sentences, paragraphs, or documents. The fundamental part of the similarity detection in a text is to perform the detection of word similarity in the document [7]. The word similarity can be either lexical or semantic similarity. Lexical similarity is rated based on the common sequence of characters in the word.

The string-based algorithm is a method used to measure lexical similarity. String-based similarity measures the similarity and dissimilarity through the order and arrangement of the characters that make up words [5]. String-based algorithms are differentiated into two groups of methods, i.e., character-based and term-based or token-based similarity [5,7].

Character-based similarity measures the similarity of two strings through the edit distance needed to make the two strings to be equal. Edit distance includes insertion, substitution, and deletion character. The similarity value is obtained based on the required minimum edit distance [5]. Some examples of methods that use a character-based similarity approach are Hamming distance, Levenshtein distance [6,8-10], Jaro, Jaro-Winkler [2,3,5,11], and N-gram.

The term-based or token-based similarity measures string similarity based on the similarity of token strings with token sets [5]. Usually, the token is in the form of a word in a string. The term-based similarity method is such as Cosine similarity, Euclidean distance, Manhattan distance, Dice's coefficient, and Jaccard similarity [5].

The character-based similarity becomes very computational and has a low level of accuracy if it is used to measure document similarity with many strings [5]. Meanwhile, the term-based similarity will be less accurate if the document has very diverse tokens and the token slice on the document is low. Based on several weaknesses of the two string-based approaches, a hybrid method was developed based on character-based and term-based similarity. One of them is SoftTFIDF.

The SoftTFIDF method is a hybrid method that combines a token-based similarity approach with character-based similarity. SoftTFIDF was developed by Cohen et al. [12].

The Hybrid SoftTFIDF approach is used in conjunction with character-based similarities such as Levenstein, Jaro-Winkler, and other approaches [11-13]. The SoftTFIDF method can work well in detecting document similarity; however, this algorithm works by performing two measurements, namely the TFIDF calculation as in the cosine algorithm and calculating the second algorithm such as Jaro-Winkler or Levenstein. The CombineTF method simplifies the calculation process on SoftTFIDF, wherein CombineTF, the first measurement, does not need to be carried out until the TFIDF calculation. Instead, it is enough to measure the TF and Document Frequency (DF) of the two strings.

3. **Proposed Method: CombineTF.** The CombineTF method is a hybrid method that combines token-based and character-based similarity. The CombineTF algorithm and flowchart can be seen in Figure 1.

Algorithm 1: CombineTF Algorithm
Input Data: string s, string t
Result: $sim_{combineTF}$
Create the set of terms w
For each pair of string (s,t) **do**
  For each term w **do**
    Compute term frequency $tf_{w,s}$
    Compute term frequency $tf_{w,t}$
    Compute document frequency df_w
    If $df_w = |D|$
      For $\min(tf_{w,s}, tf_{w,t})$ **do**
        Create metric similar term *dic*
      Enddo
    Endif
  Enddo
  Create new string *ns*
  Create new string *nt*
  For each pair of strings (ns,nt) **do**
    Compute *sim*2
  Enddo
  Compute the number of terms N
  Compute the number of terms n
  Compute similar term proportion p
  Computer similarity $sim_{combineTF}$
Enddo

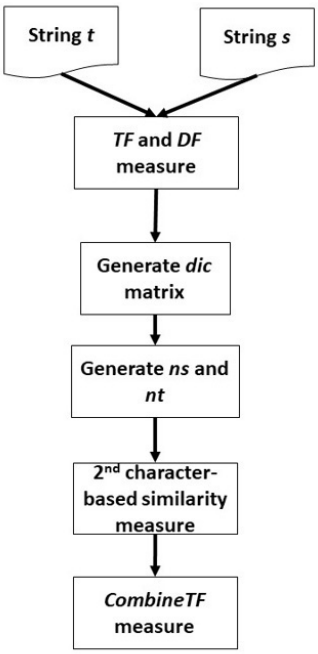

FIGURE 1. Algorithm and flowchart of CombineTF method

In Algorithm 1, it can be seen that the method accepts input in the form of string s and string t and produces output in the form of CombineTF similarity. The method works in three stages. First, it determines the same terms on the strings s and t. The first step at this stage is to concatenate strings s and t to form a set of terms w. After that, the calculation of $tf_{w,s}$ and $tf_{w,t}$ is carried out, and continued with the calculation of df_w where w is the number of frequencies of the term, the next step is to make *dic* matrix which contains a list of similar terms on s and t. Parameter $|D|$ is the number of tested documents or strings. In the testing similarity case of two strings, therefore value of $|D|$ is 2. The last step in the first stage is to form new strings such as *ns* and *nt*, where *ns* and *nt* are strings formed from s and t after subtracting *dic*. It means *ns* and *nt* have different terms from s and t. The equations used in the first stage can be seen

from Equations (1) and (2).

$$dic = \{w|df_w = |D|, \min(tf_{w,s}, tf_{w,t})\} \tag{1}$$

$$ns = s - dic \tag{2}$$

The second stage applies the calculation with character-based similarity. In this study, we use two approaches, i.e., CombineTF and Jaro-Winkler (TF-JW) or CombineTF with Levenshtein (TF-Leven). Equation (3) below is used for the process of checking the contents of the ns and nt strings where $sim2$ is a method similarity combined with TF; in this case, it is Jaro-Winkler or Levenshtein.

$$sim2 = \begin{cases} sim2_d & \text{if } ns \neq \emptyset \text{ and } nt \neq \emptyset \\ 1 & \text{if } ns = \emptyset \text{ and } nt = \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The last stage is calculating CombineTF similarity by calculating p as the same term proportion on the two strings and concatenating it with the value of $sim2$ in Equations (4)-(7).

$$N = |s| + |t| \tag{4}$$

$$n = |ns| + |nt| \tag{5}$$

$$p = \frac{(N - n)}{N} \tag{6}$$

$$sim_{combineTF} = (1 \times p) + (sim2 \times (1 - p)) \tag{7}$$

where N is the number of terms on s and t strings, and n is the number of terms on ns and nt. Parameter p is the same term proportions on s and t, and $sim_{combineTF}$ is the summation of the same terms on both strings and $sim2$ different term similarity from both strings.

For example, suppose string s = "member data processing", and string t = "member status data". Based on data on TF and DF value, we obtain dic = {member, data}, so that ns = "processing" and nt = "status". Based on the calculation, we obtain $sim2$ (in this case is Jaro-Winkler) from ns and nt is 0.4892. With the sum of all terms six and the number of different terms being 2, therefore, we obtain $N = 6$ and $n = 2$, so we can calculate $p = 0.6667$. Based on the values of all the variables, we obtain $sim_{combineTF} = 0.8297$.

4. **Experiment and Result.** The experiment is carried out through five stages, starting with the data collection of the stakeholder's requirements, continuing with the preprocessing. Therefore, a requirements dataset in which the similarity will be measured is formed. Next, we do the labeling process for the requirements dataset. Finally, we do the requirements similarity measurement. The experiment stages can be seen in Figure 2.

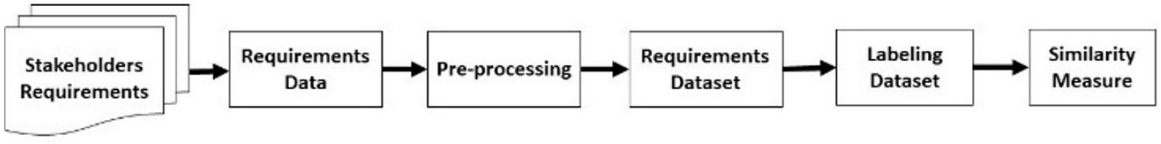

FIGURE 2. The experiment stages

4.1. **Requirements data in AREM.** AREM is a model for automating the requirements engineering process. This model uses a Goal-Oriented Requirements Engineering (GORE) approach. In GORE, there are several elements that represent user requirements, such as goals, tasks, and operations. The requirements data in AREM are in the form of descriptions of each element in GORE in Bahasa Indonesia. Each element becomes an

object in the model. Table 1 presents an example of data on the requirements of stakeholders for a cooperative information system. The requirements data is a combination of all elements of the model and is a combination of all stakeholders. The requirements data that have been collected will be pre-processed in the form of case folding and stopword removal.

Table 1. Example of requirements data

Description	Object type
Make it easier to display information about cooperative	goal
Calculating cooperative profit	task
Record member installment payments	operational

4.2. **Requirements datasets and labeling.** The requirement dataset is formed from the requirements data by pairing each object to measure the similarity of the two objects. After forming the dataset, data labels are then performed for each pair of objects. Labeling is carried out by experts who assess the similarity of the descriptions of the two objects. Labeling is done by assigning a number 1 for both descriptions of the same object and 0 for different ones. Table 2 shows an example of a requirements dataset.

Table 2. Example of requirements dataset

Object description I	Object description II	Label
Collecting member loan data	Calculating cooperative profit	0
Collecting member loan data	Calculating member's loan	1
Collecting member loan data	Record member installment payments	0

For the needs of the experiment, eight datasets were formed. The datasets are divided into two types of data, namely balanced datasets, and unbalanced datasets. The composition for the unbalanced dataset is 20% of the data, it has a label of 1, and the rest is labeled 0 for datasets 5, 6, and 7. Meanwhile, for dataset 8, the opposite applies where 20% of the data is labeled 0 and 80% is labeled 1. Each type of dataset is divided into four data groups with a different number of descriptive sentences. The first group consists of one descriptive sentence, the second group has three descriptive sentences in each description, the third group has five sentences, and the last group has ten sentences for each description. Table 3 presents a group of datasets used in the experiment.

Table 3. Experiment dataset

No	Dataset type	Number of sentences	Number of data	No	Dataset type	Number of sentences	Number of data
1	Balance	1	1,128	5	Unbalance	1	2,820
2	Balance	3	1,128	6	Unbalance	3	2,820
3	Balance	5	2,256	7	Unbalance	5	5,640
4	Balance	10	736	8	Unbalance	10	1,440

4.3. **Similarity measure.** In the experiment, the similarity measurement was conducted using six methods, that is, Cosine similarity, Jaro-Winkler, Levenshtein distance, Soft-TFIDF with Jaro-Winkler, CombineTF with Jaro-Winkler (TF-JW), and CombineTF with Levenshtein Distance (TF-Leven). Cosine similarity is one of the methods from the term-based similarity approach, while Jaro-Winkler and Levenshtein distance are methods from the character-based similarity approach. Meanwhile, SoftTFIDF is one of the

approaches which are developed based on TF-IDF calculation [12,13]. This method is often combined with Jaro-Winkler as a character-based similarity approach. In the implementation of the Jaro-Winkler Algorithm, we use value for the parameter $b_t = 0.7$, $p = 0.1$, and $l \leq 4$ [3].

4.4. **Result and discussion.** The result analysis was done to the average precision and recall, F1 score with the dataset, analysis of F1 score with the threshold, analysis of average F1 score, and average F1 score, particularly for the CombineTF method. The test was done for eight datasets with six methods and a threshold from 0 to 1.

The results of precision and recall analysis show that the average precision and recall (as shown in Figure 3) from the six algorithms gives almost the same results, except for the Jaro-Winkler algorithm, which shows lower relation between precision and recall than other algorithms. CombineTF-JW is able to improve precision and recall in the Jaro-Winkler algorithm.

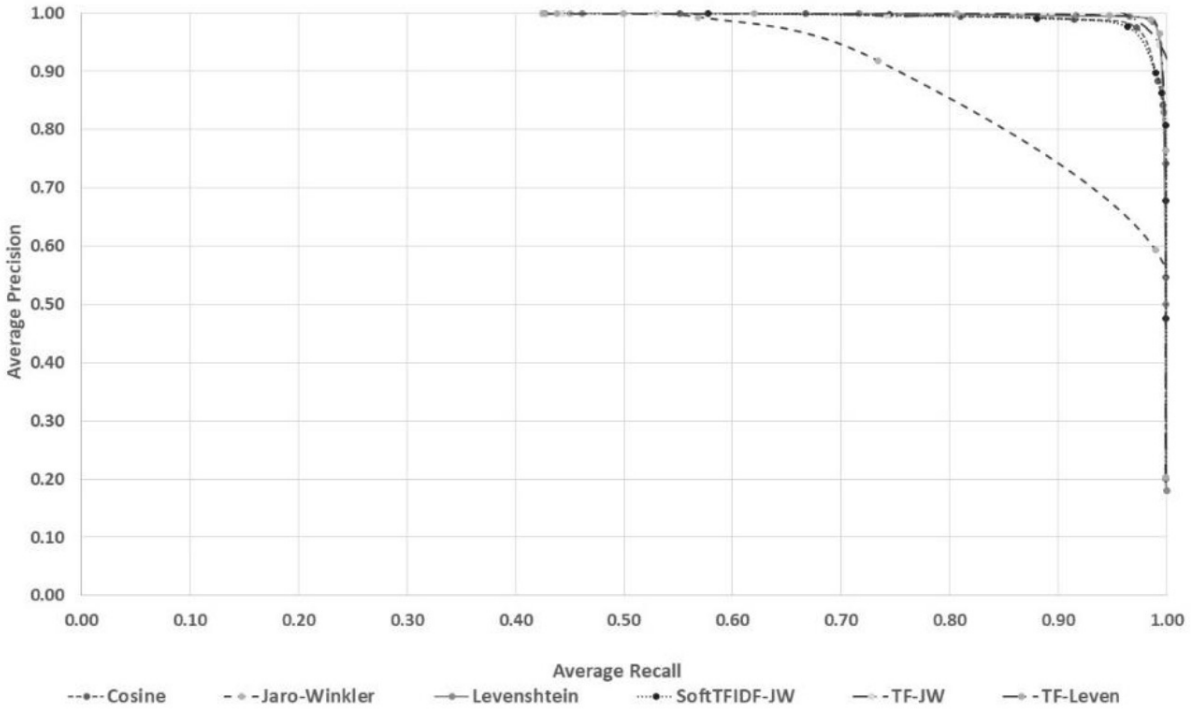

FIGURE 3. Analysis of average precision and recall of the six methods tested

The results of average F1 scores for each dataset can be seen in Figure 4. Based on the distribution of average F1 scores in Figure 4, it can be seen that the Cosine and SoftTFIDF-JW algorithms have the highest distribution and maximum F1 score, followed by Levenshtein and TF-Levenshtein. Meanwhile, CombineTF-JW and Jaro-Winkler have the lowest maximum F1 score.

If we analyze from the type of the dataset, it can be seen that all algorithms show the same trend, such as decreasing from datasets 1 to 7. However, it is interesting on dataset 8, the average F1 score of all algorithms reaches the highest value from the entire dataset. Dataset 8 is a dataset that contains unbalanced data where 80% of the data is labeled 1. Meanwhile, datasets 5 to 7 contain unbalanced data with 80% labeled as 0. This means that all algorithms work better in testing similar data compared to dissimilar data. While the downward trend in datasets 1 to 4 indicates that the more sentences in a pair of strings s and t, the F1 score of the algorithm also decreases.

F1 score analysis was also carried out by looking at the threshold value applied to the experiment. Overall, the average F1 measure which was measured by the threshold,

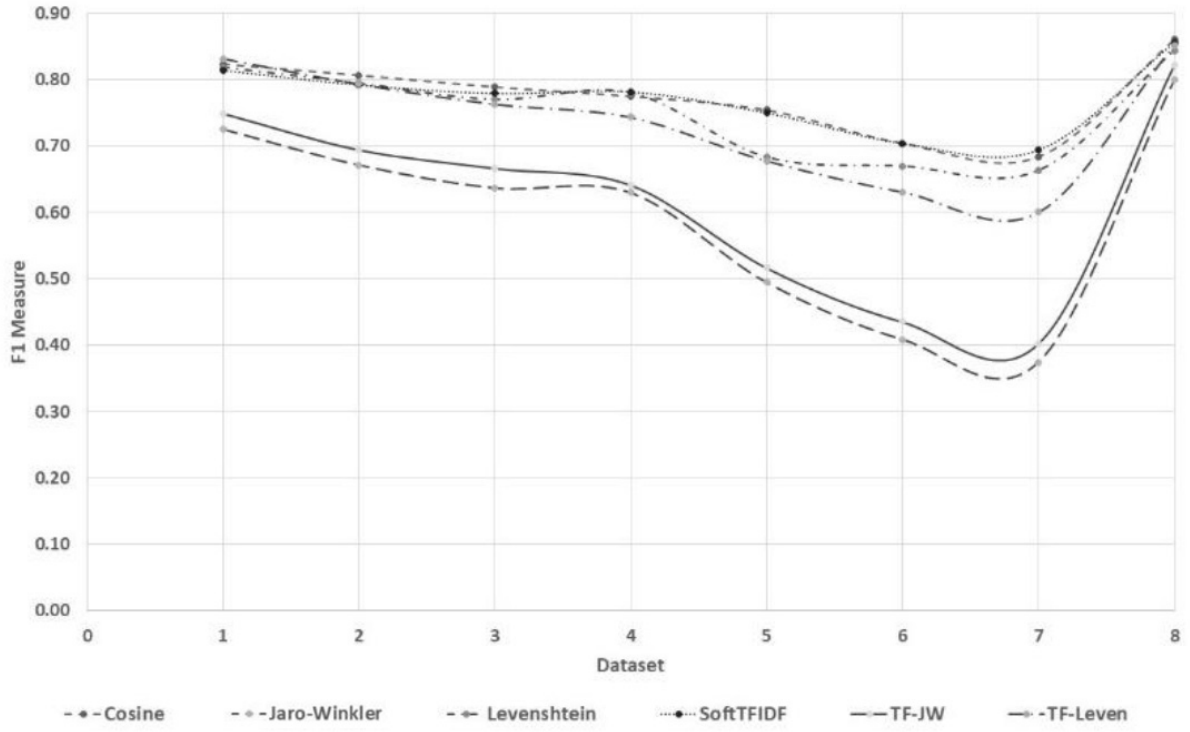

FIGURE 4. Trend analysis of average F1 score and dataset

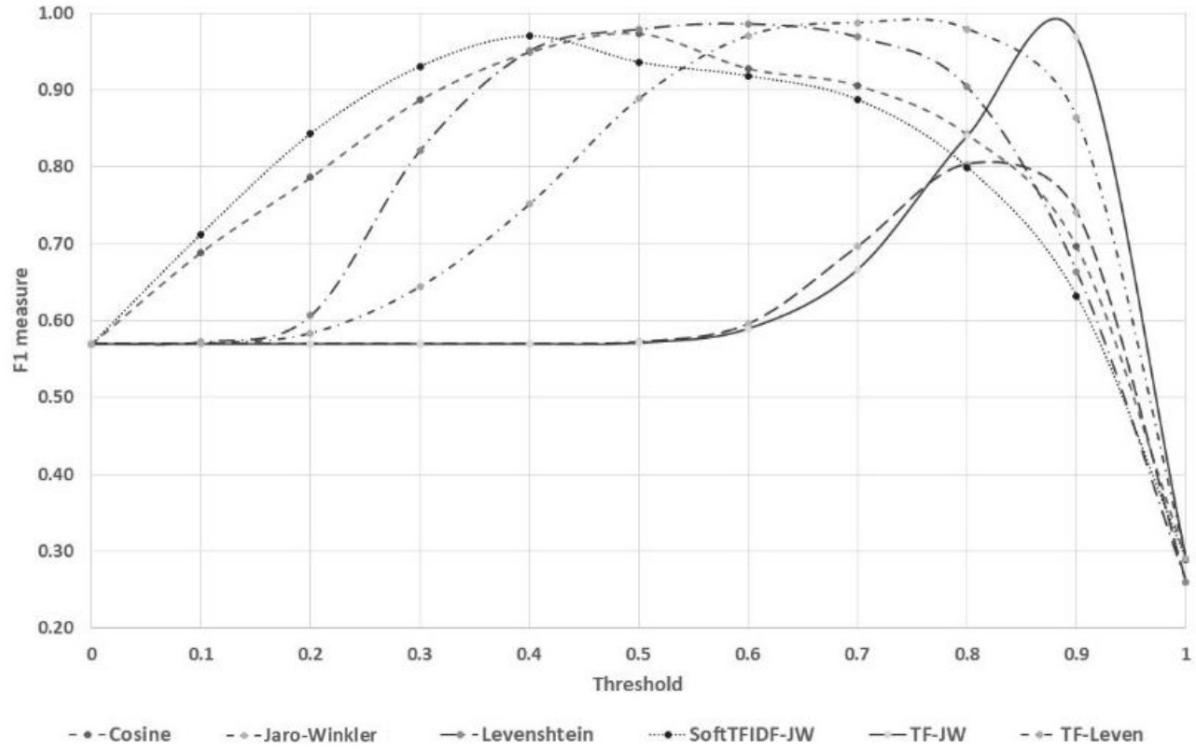

FIGURE 5. Trend analysis of average F1 score and threshold

shows a trend, as shown in Figure 5. It can be seen in Figure 5, the cosine algorithm has a high F1 score at a threshold of 0.5 to 0.7, while SoftTFIDF-JW and Levenshtein distance reach the highest F1 score at the threshold of 0.4 to 0.7. The maximum shift in F1 score is shown in CombineTF-Levenshtein, wherein Levenshtein falls on a threshold of 0.4 to 0.7, then in CombineTF-Levenshtein F1 highest score is achieved at a threshold of 0.6 to 0.8. Jaro-Winkler has the lowest maximum F1 score compared to other algorithms, but

CombineTF-JW is able to increase the highest F1 score in the Jaro-Winkler algorithm, where the maximum F1 score is achieved at a threshold of 0.9. Overall, the CombineTF algorithm works very well at a threshold of more than 0.5. Particularly for CombineTF-JW, it works very well at a threshold of more than 0.8.

5. **Conclusion.** In this study, the CombineTF method was developed to detect the similarity of the requirements data from stakeholders in AREM. The CombineTF is a hybrid method that combines a team-based approach with character-based similarity. The method begins with the measurement process for TF and DF, followed by measurements using the character-based method as the second method. In the experiment, two methods were used, namely Jaro-Winkler and Levenshtein distance. The trial was carried out using eight datasets that had different characteristics and amounts of data. As a comparison, the tests were carried out on four other methods besides CombineTF, namely Cosine similarity, Jaro-Winkler, Levenshtein distance, and SoftTFIDF with Jaro-Winkler.

The experimental results show that CombineTF provides better performance if the threshold for data is more than 0.5. Particularly for CombineTF-JW, it shows the best performance at a threshold greater than 0.8. For the type of dataset, CombineTF shows good performance in measuring the requirement data similarity. On the other hand, poor performance is shown when measuring dissimilarities in the dataset.

The requirements data in AREM contains a brief description of each element of the model. In the model, the main capability needed is to test the similarity of the requirements data. Based on these conditions, it can be said that CombineTF is very suitable to be used to check the requirement data similarity between stakeholders in AREM.

In this research, the experiment was limited to the requirements data in AREM. To be able to find out the performance of the CombineTF method more generally, it is necessary to test the method on general text documents such as news documents or other documents. For method development, it is possible to add the N-Gram approach as done by [14] as a second approach to measure the similarity of text that has different terms/tokens.

Acknowledgment. The authors thank the Republic of Indonesia Ministry of Research and Higher Education through Universitas Gadjah Mada, which has funded this research with contract numbers: 2271/UN1/DITLIT/DIT-LIT/PT/2021.

REFERENCES

[1] C. D. Manning, P. Raghavan and H. Schütze, *Introduction to Modern Information Retrieval*, 2nd Edition, Cambridge University Press, 2009.

[2] M. A. Jaro, Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida, *J. Am. Stat. Assoc.*, vol.84, no.406, pp.414-420, DOI: 10.1080/01621459.1989.10478785, 1989.

[3] W. E. Winkler, *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*, http://files.eric.ed.gov/fulltext/ED325505.pdf, 1990.

[4] S. C. Cahyono, Comparison of document similarity measurements in scientific writing using Jaro-Winkler Distance method and Paragraph Vector method, *IOP Conference Series: Materials Science and Engineering*, vol.662, no.5, pp.1-9, DOI: 10.1088/1757-899X/662/5/052016, 2019.

[5] D. D. Prasetya, A. P. Wibawa and T. Hirashima, The performance of text similarity algorithms, *Int. J. Adv. Intell. Informatics*, vol.4, no.1, pp.63-69, DOI: 10.26555/ijain.v4i1.152, 2018.

[6] M. M. Yulianto, R. Arifudin and A. Alamsyah, Autocomplete and spell checking levenshtein distance algorithm to getting text suggest error data searching in library, *Sci. J. Informatics*, vol.5, no.1, p.75, DOI: 10.15294/sji.v5i1.14148, 2018.

[7] W. H. Gomaa and A. A. Fahmy, A survey of text similarity approaches, *Int. J. Comput. Appl.*, vol.68, no.13, pp.13-18, DOI: 10.5120/11638-7118, 2013.

[8] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, *Cybernatics Control Theory*, vol.10, no.8, pp.707-710, DOI: 10.1016/S0074-7742(08)60036-7, 1966.

[9] G. Huang, J. Chen and Z. Sun, A correction method of word spelling mistake for English text, *J. Phys. Conf. Ser.*, vol.1693, no.1, pp.1-7, DOI: 10.1088/1742-6596/1693/1/012118, 2020.

[10] K. N. S. Behara, A. Bhaskar and E. Chung, A novel approach for the structural comparison of origin-destination matrices: Levenshtein distance, *Transp. Res. Part C Emerg. Technol.*, vol.111, no.11, pp.513-530, DOI: 10.1016/j.trc.2020.01.005, 2020.

[11] Y. van Gennip, B. Hunter, A. Ma, D. Moyer, R. de Vera and A. L. Bertozzi, Unsupervised record matching with noisy and incomplete data, *Int. J. Data Sci. Anal.*, vol.6, no.2, pp.109-129, DOI: 10.1007/s41060-018-0129-7, 2018.

[12] W. W. Cohen, P. Ravikumar and S. E. Fienberg, A comparison of string metrics for matching names and records, *American Association for Artificial Intelligence*, 2003.

[13] N. Gali, R. Mariescu-Istodor and P. Fränti, Similarity measures for title matching, *Proc. of International Conference on Pattern Recognition*, pp.1548-1553, DOI: 10.1109/ICPR.2016.7899857, 2016.

[14] D. D. Sinaga and S. Hansun, Indonesian text document similarity detection system using Rabin-Karp and Confix-Stripping algorithms, *International Journal of Innovative Computing, Information and Control*, vol.14, no.5, pp.1893-1903, DOI: 10.24507/ijicic.14.05.1893, 2018.