The Journal of Supercomputing (2024) 80:363–394
https://doi.org/10.1007/s11227-023-05472-0

JCF: joint coarse- and fine-grained similarity comparison for plagiarism detection based on NLP

Chih-Yung Chang[1] · Syu-Jhih Jhang[1] · Shih-Jung Wu[1] · Diptendu Sinha Roy[2]

Accepted: 1 June 2023 / Published online: 24 June 2023

Abstract

Document similarity recognition is one of the most important problems in natural language processing. This paper proposes a plagiarism comparison mechanism called *JCF*. Initially, the TF–IDF scheme is applied to build a bag of words as the representation of the common features of all documents. Then, the plagiarism comparison is carried out in a coarse-grained manner, which speeds up the similarity comparison. Finally, the most similar documents can then be compared in detail based on a fine-grained approach. In addition, the *JCF* detects plagiarism at both syntax level and semantic-like level. To prevent the distortion of similarity comparison, this paper further develops a similarity restoration approach such that the proposed *JCF* can obtain both advantages of quickness and accuracy. Performance studies confirm that the proposed *JCF* outperforms existing studies in terms of precision, recall and F1 score.

Keywords Natural language processing · TF–IDF · Word2Vec · Coarse and fine grained · Document similarity

✉ Chih-Yung Chang
cychang@mail.tku.edu.tw

Syu-Jhih Jhang
810440064@gms.tku.edu.tw

Shih-Jung Wu
wushihjung@mail.tku.edu.tw

Diptendu Sinha Roy
diptendu.sr@nitm.ac.in

[1] Tamkang University, New Taipei City, Taiwan

[2] National Institute of Technology, Shillong, India

1 Introduction

Plagiarism comparison of papers is an important research topic in the field of natural language processing. Although the existing text similarity comparison system can provide users with the plagiarism comparison of sentences and paragraphs, the existing text similarity comparison methods need to go through many time-consuming processes, which compare the word similarity in a one-by-one manner between the target document and all the documents in the database. As a result, it takes a lot of time to wait for the comparison results.

In the literature, some studies have proposed similarity comparison mechanisms based on Word2Vec, N-gram or Bert. However, it is time-consuming because that one document often contains a large number of words and is required to be compared the similarity with each of a large number of documents in a manner of one-to-one comparison. In most text comparison methods, to obtain the features of the document, all documents in the document database will be segmented and sentenced, and then, the keywords of all documents will be obtained through TF–IDF [1, 2]. After that, the Word2Vec model was applied to convert the text into vectors, which were readable input formats by a computer. The Word2Vec was a method proposed by Mikolov et al. [3–7]. It mainly transferred the texts to vectors. Two mechanisms were generally applied to achieve the function of Word2Vec. The first one was a continuous bag of words (CBOW) [8, 9], which used adjacent words as the input of the neural network to predict the target word. The second one was skip-gram [10, 11], which used the target word as the input of the neural network to predict what the adjacent words are. Finally, to confirm whether or not the two words belonging to different documents were similar, the cosine similarity [12, 13] was applied.

Generally speaking, the above method mainly mapped the words in the text to the vector space. Then, the distance between the two vectors was calculated. In past research [14, 15], the similarity between two documents was compared through the above procedure to find out their similarities. However, a document usually contains a large number of words. Let the source of the comparison be a document, and the goal of the comparison be to compare thousands of documents in the database. It was time-consuming to compare the similarity of one target document and all documents in the database since the existing mechanisms only can compare the similarity of two documents at one time. As a result, plagiarism comparison is very time-consuming.

This study proposes a plagiarism comparison mechanism, called *JCF*, aiming to save time for similarity comparison between one target document and all documents in the database. The proposed *JCF* first takes out the important keywords of each document in the way of bag of words [16, 17]. These keywords will be considered as the document vector. Based on the document vectors, similar documents can be quickly identified, saving time for comparing a large number of documents. After that, the proposed mechanism further compares the sentences containing those similar keywords in similar documents. If the sentences are also similar, the proposed mechanism further compares the paragraphs containing similar sentences. Finally, the proposed *JCF* will compare the documents with

particularly similar paragraphs word by word. Different from the previous studies, this study uses a coarse-to-fine way to compare the similarity of documents, which can save a lot of comparison time.

In addition, since the similarities of word vectors, sentence vectors and paragraph vectors were used for comparison, it is almost able to check the plagiarism at the semantic level. Therefore, this study does not stick to the word-to-word ratio for its repetition, which was more flexible in terms of semantics. The main contributions of this study are itemized as follows:

1. *Speed up the similarity comparison* Most related studies compared the similarity of the word vectors of the two documents. However, when the compared targets are a set of documents, similar comparisons are time-consuming. The proposed *JCF* speeds up the similarity comparison since it applies the Bag of Words scheme to initially establish a set of common features of all documents and then transforms each document as a document vector. Then, the most similar document can be found as the candidate document. As a result, the target document only needs to be compared with the candidate document. Therefore, the proposed *JCF* has better performance in terms of comparison time.
2. *Coarse-grained and fine-grained design* The proposed *JCF* first identifies the most similar document from a set of candidate documents in a manner of coarse grain, which only compares the document vector. Then, the candidate document is compared with the target document in a manner of coarse grain, which compares the similarity in keyword level, sentence vector level and then paragraph vector. Finally, the fine-grained policy is applied to find similar words by comparing the word vectors of the two documents.
3. *Semantic-level similarity comparison* Most related studies compared the two documents using word vectors. The comparison of word vectors can reflect the semantic similarity in case the similarity of two vectors is higher than the predefined threshold. However, instead of comparing all sentences or all paragraphs, the proposed *JCF* only compares the vectors of those sentences and paragraphs which contain similar words. This can speed up the comparison of the two documents and further extract the semantic similarity at the sentence level and paragraph level.
4. *Similarity restoration* It is not possible to compare the similarity of two documents in a way of word by word since it is time-consuming. To speed up the similarity comparison, the comparison can be performed after extracting the features of the two documents. For example, TF–IDF, bag of words, sentence vector, paragraph vector or document vector is well-known feature extraction functions that have been generally applied to a document. However, when applying the feature extraction functions, the similarity comparison of two documents might occur distortion, which might impact the accuracy of plagiarism detection. The proposed *JCF* further develops a similarity restoration mechanism such that the similarity comparison can reduce much of the time and prevent plagiarism detection from distortion.

2 Related work

The core purpose of this study is to design a comparison method to quickly find similar documents. This section describes the related studies dealing with the similarity of documents in recent years. Technically, related studies can be roughly divided into three categories, including technologies based on BERT, N-gram and Word2Vec.

Rosu et al. [18] proposed a deep learning plagiarism detection method. This study collected 570,000 sentences as a dataset for training the BERT model. In this study, the contents of documents A and B were first segmented, and then, the sentences of the two documents were converted into sentence vectors through BERT. Then, each sentence of document A was compared with each sentence of document B according to the sentence vector. The sentence of document A, having the highest similarity with the one of document B, can be obtained. Finally, dividing the sum of sentences with the highest similarity by the number of sentences was the similarity of document Bohra, and Barwar. [14] used BERT as a model to propose a plagiarism detection system. This study first removed stop words from the two documents A and B and then divides the two documents into many sentences. After that, each sentence of the two documents A and B played the role of input of the sentence transformer to obtain its sentence vector. Then, the two documents were divided into sentences. The sentences of the document were compared one by one for similarity. Then, a threshold value was set to filter the sentences with high similarity. Finally, similar to Rosu et al. [18], the sentences that satisfied the threshold value were summed up and the average was obtained as the document similarity.

Yalcin et al. [19] proposed a part-of-speech tag (POS) N-gram plagiarism detection system. This study first segmented the two compared articles. Then, it removed the stop words and further segmented sentences from the articles. By tagging the processed words through POS, the part-of-speech structure of the sentence can be obtained. Then, the similarity of part-of-speech structures of the two articles was compared. If they were similar, it further compared the sentence part-of-speech structure of the whole article. If the similarity of the sentences was higher than the specified value, the sentence similarity is compared through Word2Vec, and finally, the comparison results of similar sentences were displayed. Awale, et al. [20] proposed an approach to detect plagiarism in programming assignments by using *N*-grams and machine learning techniques. The method involved extracting features from source code using *N*-grams and then training a classifier using various machine learning algorithms to identify similarities between source codes. The source code underwent preprocessing, including removing comments, whitespace and other irrelevant symbols. The method extracted *N*-gram features from the preprocessed source code using the sliding window technique. The method trained a classifier using several machine learning algorithms, including random forest, support vector machine and multilayer perceptron, aiming to distinguish between plagiarized and non-plagiarized programming assignments. The classifier was trained on the extracted *N*-gram features.

The proposed approach computed the similarity score between the feature sets of the source and suspicious documents using the cosine similarity measure.

Ramadhanti et al. [21] proposed a method for similarity comparison of Indonesian articles based on the Word2Vec model. The research first collected a large number of Indonesian articles through Wikipedia and then used the collected articles as a training set as the input data of the Word2Vec model to train word vectors. Then, the similarity was checked based on two parts: the document similarity comparison and the paragraph similarity comparison. In the document comparison, all the words in the document were transferred into word vectors using Word2Vec. Then, the word vectors of all the words in the two articles were compared using the Cosine similarity method. Similarly, the paragraph similarity comparison aimed to compare the paragraphs of the two articles. It firstly obtained the word vectors of all words in the two paragraphs through Word2Vec, and it used the Cosine similarity to calculate the word vectors of the paragraphs. Finally, the similarity of articles and paragraphs can be obtained. Xia et al. [15] proposed a similarity comparison method based on the Word2Vec model. In this study, the two law documents were first segmented, and then, the sentence vectors were obtained through Word2Vec. After that, the sentences of the two articles were compared one by one for similarity. The similarity comparison method mainly used the Cosine similarity and Word Movers Distance methods. Finally, the sentences with the highest similarity were summed and divided by the total number of sentences as the similarity of the document. Qurashi et al. [22] proposed a similarity comparison method for railway regulatory documents based on the Word2Vec model. It collected railway regulatory documents through Google News and used the collected documents as a training set. Initially, it segmented the two railway regulations documents. Then, it inputs the two documents into the Word2Vec model to obtain the sentence vectors of all sentences. After that, it compared the sentences in the documents one by one. The similarity comparison method used Cosine similarity and Jaccard similarity for calculation. After obtaining the highest similarity of each sentence, the similarity of all sentences is summed and divided by the total number of sentences to represent the document similarity.

The above-mentioned studies are based on BERT, *N*-gram and Word2Vec. When comparing the similarity, most of them are a single comparison of sentences, paragraphs or articles. However, when the similarity comparison aims to compare one article with a set of tens of thousands of articles, the comparisons will be time-consuming. In case there are multiple articles needed to be compared, it takes a long time and cannot be compared quickly. Table 1 compares the aforementioned seven documents with our study. The comparison includes examining the target paper alongside multiple papers, evaluating similarity error restoration, and analyzing the progression from coarse to fine. In the table, the notation "V" indicates the matching for the criterion while the notation "-" indicates its absence.

Table 1 Comparison of related studies

Related work	One-to-many comparison	Model	Similarity error restoration	From coarse to fine
[18]	–	BERT + Cosine similarity	–	–
[14]	–	BERT + Cosine similarity	–	–
[19]	–	N-gram + POS + Word2Vec	–	v
[20]	–	N-gram + Cosine similarity	–	–
[21]	–	Word2Vec + Cosine similarity	–	v
[15]	–	Word2Vec + Cosine similarity + WMD	–	–
[22]	–	Word2Vec + Cosine similarity + Jaccard similarity	–	–
Our	v	Word2Vec + Cosine similarity	v	v

3 Assumptions and problem formulation

This section introduces the assumptions and the problem statements. This paper proposes the document similarity comparison mechanism which finds the most similar document from the database for a given new document. Let $U = \left\{D^1, D^2, \ldots, D^q\right\}$ denote the set of q documents. Each document $D^i = \{P^{i,1}, P^{i,2} \ldots, P^{i,|D^i|}\}$ is composed of several paragraphs $P^{i,j}$, where $1 \le j \le \left|D^i\right|$. Each paragraph $P^{i,j} = \{s^{i,j,1}, s^{i,j,2}, \ldots, s^{i,j,|s^{i,j}|}\}$ is composed of several sentences $s^{i,j,k}$, where $1 \le k \le \left|P^{i,j}\right|$. Each sentence $s^{i,j,k} = \{w^{i,j,k,1}, w^{i,j,k,2}, \ldots w^{i,j,k,q}\}$ is composed of several words in the sentence $s^{i,j,k}$. Given a document D^{target}, this paper aims to develop a document comparison mechanism, which compares D^{target} and each document D^i in D and finds the most similar document. Let $D^{\text{target}} = \left\{p^{t,1}, p^{t,2} \ldots, p^{t,|D^{\text{target}}|}\right\}$ be composed of several paragraphs $P^{t,\widehat{j}}$, where $1 \le \widehat{j} \le \left|D^i\right|$. Let $p^{t,\widehat{j}} = \{s^{t,\widehat{j},1}, s^{t,\widehat{j},2}, \ldots, s^{t,\widehat{j},\left|p^{t,\widehat{j}}\right|}\}$ be composed of several sentences $s^{t,\widehat{j},\widehat{k}} \in p^{t,\widehat{j}}, 1 \le \widehat{k} \le \left|P^{i,j}\right|$. $s^{t,\widehat{j},\widehat{k}} = \{w^{t,\widehat{j},\widehat{k},1}, w^{t,\widehat{j},\widehat{k},2}, \ldots w^{t,\widehat{j},\widehat{k},\left|s^{t,\widehat{j},\widehat{k}}\right|}\}$ be composed of several words $w^{t,\widehat{j},\widehat{k},\widehat{q}} \in s^{t,\widehat{j},\widehat{k}}, 1 \le \widehat{q} \le \left|s^{t,\widehat{j},\widehat{k}}\right|$. Consider two words $w^{i,j,k,q} \in D^i$ and $w^{t,\widehat{j},\widehat{k},\widehat{q}} \in D^{\text{target}}$. Let $\lambda^{i,j,k,q}_{t,\widehat{j},\widehat{k},\widehat{q}}$ be a Boolean variable that indicates whether or not the word $w^{i,j,k,q} \in D^i$ is identical to the word $w^{t,\widehat{j},\widehat{k},\widehat{q}} \in D^{\text{target}}$. That is,

$$\lambda^{i,j,k,q}_{t,\widehat{j},\widehat{k},\widehat{q}} = \begin{cases} 1 & w^{i,j,k,q} = w^{t,\widehat{j},\widehat{k},\widehat{q}} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

If the condition $\lambda^{i,j,k,q}_{t,\widehat{j},\widehat{k},\widehat{q}} = 1$ holds, it represents that q-th word in the document D^i is the same as the $\widehat{q}$-th word in the document D^{target}.

The next step is to compare the k-th sentence $s^{i,j,k}$ in document D^i and the $\widehat{k}$-th sentence $s^{t,\widehat{j},\widehat{k}}$ in D^{target}. Let $\lambda^{i,j,k}_{t,\widehat{j},\widehat{k}}$ denote the number of identical words in the two sentences $s^{i,j,k}$ and $s^{t,\widehat{j},\widehat{k}}$. That is,

$$\lambda^{i,j,k}_{t,\hat{j},\hat{k}} = \sum_{\hat{q}=1}^{\left|s^{t,\hat{j},\hat{k}}\right|} \sum_{q=1}^{\left|s^{i,j,k}\right|} \lambda^{i,j,k,q}_{t,\hat{j},\hat{k},\hat{q}} \tag{2}$$

If the condition $\lambda^{i,j,k}_{t,\widehat{j},\widehat{k}} \geq \left|s^{i,j,k}\right|$, the paragraphs $p^{i,j}$ and $p^{t,\widehat{j}}$ that contain $s^{i,j,k}$ and $s^{t,\widehat{j},\widehat{k}}$, respectively, should be further examined. Let $\lambda^{i,j}_{t,\widehat{j}}$ denote the number of identical sentences in the two paragraphs $p^{i,j}$ and $p^{t,\widehat{j}}$. That is,

$$\lambda^{i,j}_{t,\hat{j}} = \sum_{\hat{k}=1}^{\left|p^{t,\hat{j}}\right|} \sum_{k=1}^{\left|p^{i,j}\right|} \lambda^{i,j,k}_{t,\hat{j},\hat{k}} \tag{3}$$

If the condition $\lambda^{i,j}_{t,\widehat{j}} \geq \left|P^i\right|$, let λ^i_t denote the number of identical paragraphs in the two documents D^i and D^{target}. That is,

$$\lambda^i_t = \sum_{\hat{j}=1}^{\left|D^{\text{target}}\right|} \sum_{j=1}^{\left|D^i\right|} \lambda^{i,j}_{t,\hat{j}} \tag{4}$$

Let D^{like} be the subset of D. The document $D^i \in D$ which satisfies the condition

$$\lambda^i_t \geq \delta \tag{5}$$

will be collected in D^{like}. That is,

$$D^{\text{like}} = \left\{ D^i \middle| \lambda^i_t \geq \delta \right\} \tag{6}$$

Let A be an algorithm and the set of D^{like} found by algorithm A is called D^{like}_A. Let ρ^i_A denote whether or not the document D^i is considered as the element of D^{like} by applying algorithm A. That is,

$$\delta^i_A = \begin{cases} 1, & D^i \in D^{\text{like}}_A \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Let TP_i, TN_i, FP_i and FN_i denote the True Positive, True Negative, False Positive and False Negative of the prediction result of algorithm A for the document D^i, respectively. We have

$$\begin{aligned}
\mathrm{TP}_i &= \delta^i \times \delta_A^i, \\
\mathrm{TN}_i &= (1-\delta^i) \times (1-\delta_A^i), \\
\mathrm{FP}_i &= \left(1-\delta^i\right) \times \delta_A^i, \text{and} \\
\mathrm{FN}_i &= \delta^i \times (1-\delta_A^i)
\end{aligned} \tag{8}$$

Let TP_A, TN_A, FP_A and FN_A denote the True Positive, True Negative, False Positive and False Negative of the prediction results for all document $D^i \in D$, for $1 \le i \le q$, respectively. We have

$$\begin{aligned}
\mathrm{TP}_i &= \delta^i \times \delta_A^i, \\
\mathrm{TN}_i &= (1-\delta^i) \times (1-\delta_A^i), \\
\mathrm{FP}_i &= \left(1-\delta^i\right) \times \delta_A^i, \text{ and} \\
\mathrm{FN}_i &= \delta^i \times (1-\delta_A^i)
\end{aligned} \tag{9}$$

Let $\mathcal{A}_A$, $\wp_A$ and $\mathcal{R}_A$ denote the *Accuracy*, *Precision* and *Recall* of the predictions for applying algorithm A to all documents $D^i \in D$. The values of $\mathcal{A}$, $\wp$ and $\mathcal{R}$ can be further derived by applying the following Exps. (6), (7) and (8), respectively.

$$\mathcal{A}_A = \frac{\mathrm{TP}_A + \mathrm{TN}_A}{\mathrm{TP}_A + \mathrm{TN}_A + \mathrm{FP}_A + \mathrm{FN}_A} \tag{10}$$

$$\wp_A = \frac{\mathrm{TP}_A}{\mathrm{TP}_A + \mathrm{FP}_A} \tag{11}$$

and

$$\mathcal{R}_A = \frac{\mathrm{TP}_A}{\mathrm{TP}_A + \mathrm{FN}_A} \tag{12}$$

As a result, the *F1-Score*, denoted by $\mathcal{F}_A$, is used to adjust the weights of false positives and false negatives. Exp. (9) gives the calculation of $\mathcal{F}_A$.

$$\mathcal{F}_A = \frac{2\left(\mathrm{Precision}_A \times \mathrm{Recall}_A\right)}{\mathrm{Precision}_A + \mathrm{Recall}_A} \tag{13}$$

For a given document D^{target}, let $\mathbb{A}$ denote the set of all possible mechanisms each of which can determine the most similar document D^{target}. This paper aims to develop the best algorithm $A \in \mathbb{A}$ which can minimize the number of errors and maximize the F1-score metric. The objective function of this paper can be expressed by the Exp. (10).

Objective:

$$\max\left(\frac{2\left(\mathrm{Precision}_A \times \mathrm{Recall}_A\right)}{\mathrm{Precision}_A + \mathrm{Recall}_A}\right) \tag{14}$$

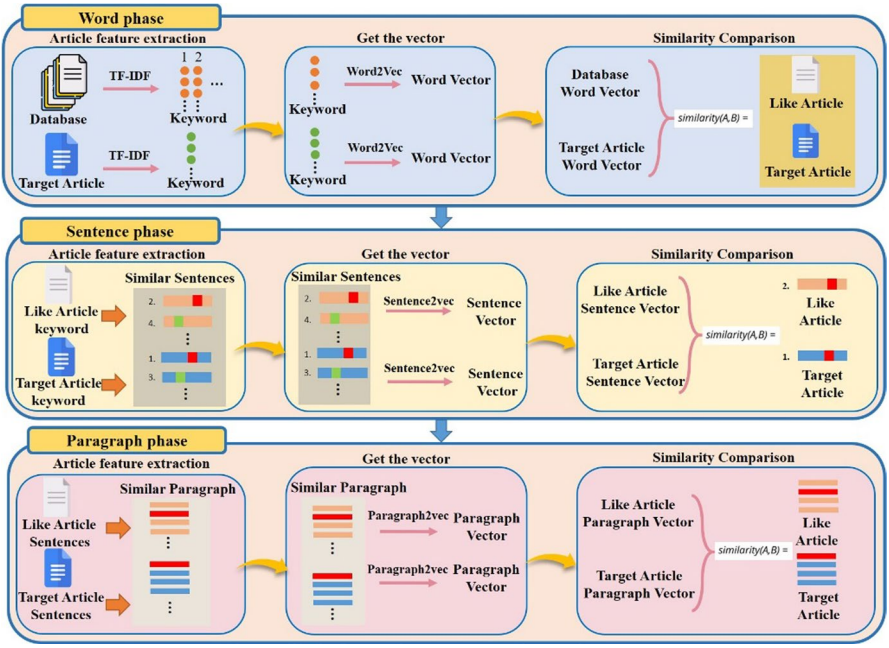

Fig. 1 An example concept flowchart of *JCF*

4 The proposed JCF mechanism

Given a documentD^{target}, this paper aims to propose a document similarity comparison mechanism that can find the most similar target document D^{like} from a given document database. Suppose the database has q documents. Let U denote the union of q documents. That is,$U = \left\{D^1, D^2, \ldots, D^q\right\}$. Let D^q become the target document D^{target}. Given a target documentD^q, this section aims to find the document D^{like} which is the document most similar toD^q. Figure 1 shows an example concept flowchart of *JCF*. The proposed *JCF* speeds up the similarity comparison since it applies the Bag of Words scheme to initially establish a set of common features of all documents. Then, it transforms each document into a document vector. As a result, the most similar document which will play the role of the candidate document can be found. As a result, the target document only needs to be compared with the candidate document.

Till now, the similar words of the two documents have been identified. Then, the comparison of the two documents in word level has been finished. The next step is to compare the two documents in sentence level. In the sentence-level comparison, only the sentences that contain similar words will be compared using the sentence vector. For two paragraphs in the target and candidate documents, if the number of similar sentences in the two paragraphs exceeds a predefined threshold, the paragraph-level comparison will be activated. The comparison of this level aims to compare two paragraphs of the target and candidate documents using paragraph vectors.

4.1 Coarse grain: similar document identification phase

Let $\widehat{w}$ denote all words in q documents. That is,

$$U = \bigcup_{i=1}^{q} D^i = \left(\widehat{w}_1^U, \widehat{w}_2^U \ldots, \widehat{w}_{|U|}^U\right) \tag{15}$$

Firstly, the TF–IDF will be applied to find the top-k important words w in U. Let w be any word in U. Let $TF\text{–}IDF(w)$ be the TF–IDF value of the word $w \in U$. Let $\widehat{U}$ be ordered $TF\text{–}IDF(w)$. Let $TF\text{–}IDF(k, \widehat{U})$ denote the set of top k words. The following gives an example to show the TF–IDF computations. Consider a document D which consists of 1000 words in total. Assume that 'AI' and 'and' are two words that appeared in document D. Firstly, the TF values of 'AI' and 'and' will be calculated. Assume that "AI" appeared 60 times, while "is" appeared 60 times in D. Let $TF(w)$ denote the TF value of word w in document D. Then.

TF("AI") = 40/1000 = 0.04, and
TF("is") = 60/1000 = 0.06.

Assume that the number of total documents considered for calculating the IDF values is 100. Assume that "AI" appears in 10 documents, and "is" appears in 100 documents. Let $IDF(w)$ denote the IDF value of the word w. Then.

IDF("AI") = log(100/10) = 1, and
IDF("is") = log(100/100) = 0,

Therefore, we have:

$TF\text{–}IDF$("AI") = TF("AI")*IDF("AI")=0.06
$TF\text{–}IDF$("is") = TF("is")*IDF("is")=0

Therefore, the importance of the word "AI" is more significant than the word "is."

Let V^U denote the set of all TF–IDF values of words $\widehat{w}_i^U$ in U. Let $top(V^U, k)$ denote the top k TF–IDF values in V^U. Let $\widehat{W}_k^U$ be the set of words whose TF–IDF values are on $top(V^U, k)$. That is, $\widehat{W}_k^U = \left\{w_i | \text{TF} - IDF\left(w_i\right) \in top(V^U, k), w_i \in U\right\}$. These words will be treated as the *document features*(or DF in short) which will be used to represent the document vector. For instance, $\widehat{W}_{10000}^U$ denote the set of 10,000 words whose TF–IDF values rank in the top 10,000.

Similarly, Let V^D denote the set of all TF–IDF values of all words w in document $D \in U$. Let $top(V^D, p)$ denote the top p TF–IDF values in V^D. Let $\widehat{W}_p^D$ be the set of words whose TF–IDF values are on $top(V^D, p)$. That is,

$$\hat{W}_p^D = \left\{w_i | \text{TF - IDF}|\left(w_i\right) \in \text{top}\left(V^D, p\right)\right\}. \tag{16}$$

For instance, let $p = 100$. W_{100}^D denote the most important 100 words in document D.

Let *reorder()* reorder all words in a set such that the subscript index of all words is continuous. Let

$$\begin{aligned} W_k^U &= reorder\left(\hat{W}_k^U\right), \text{ and} \\ W_p^D &= reorder\left(\hat{W}_p^D\right) \end{aligned} \tag{17}$$

4.2 Fine grain: word similarity check phase

Till now, the W_k^U represents the most important top-k words in all documents $D^i \in U$ and $W_p^{D^i}$ represents the top-p most important words in the document $D^i \in U$. The next step aims to create a document vector for each document $D^i \in U$. Let $\Phi(D)$ be a k-dimensional vector

$$\Phi(D) = \left(d_1, \ldots, d_k\right) \tag{18}$$

that denotes the document vector of D. Each element d_i in $\Phi(D)$ is a pair of (ϕ_i,w_i), where ϕ_i is a value denoting the similarity of word $w_i \in D$ similar to the i-th word in $\widehat{W}_k^U$. Let w_i and w_i^U be the i-th words of D and $\widehat{W}_k^U$, respectively. Let v_i and v_i^U be the word vectors of w_i and w_i^U obtaining from Word2Vec algorithm. The ϕ_i can be calculated by

$$\phi_i = \mathrm{Cos}\left(v_i, v_i^U\right) = \frac{\sum_{i=1}^{n} v_i v_i^U}{\sqrt{\sum_{i=1}^{n} {v_i}^2}\sqrt{\sum_{i=1}^{n} {v_i^U}^2}} \tag{19}$$

Let $\Phi(D).sim$ and $\Phi(D).word$ be the two vectors consisting of all similarity values ϕ_i and all words in$\Phi(D)$, respectively. For example, the document vector $\Phi(D)$ is:

$$\Phi(D) = \left(\left[0.8, 'better'\right], \left[0, 'mouse'\right], \left[0.16, 'future'\right], \ldots\right) \tag{20}$$

Then, we have

$$\begin{aligned} \Phi(D).sim &= (0.8,\ 0,\ 0.16,\ \ldots) \text{ and} \\ \Phi(D).word &= \left('\text{better}', '\text{mouse}', '\text{future}', \ldots\right) \end{aligned} \tag{21}$$

Let $\Phi(D,i).sim$ and $\Phi(D,i).word$ denote the i-th element in $\Phi(D).sim$ and$\Phi(D).word$, respectively. For example, $\Phi(D,2).sim$ is (0.16) and $\Phi(D,2).word$ ='future'. The $\widehat{W}_k^U$ containing k-word which can be treated as a k-dimensional word. At the conceptual level, each word w in $\widehat{W}_p^D$ will compare its similarity with each word in $\widehat{W}_k^U$ and finally obtain k similarity values. The k similarity values will be considered as the document vector of document$D$. The detailed operation for creating the document vector will be presented below. Let $w_{i,j}$ be the j-th word in$\widehat{W}_p^D$. Let w_{max}^U be the most similar word in $\widehat{W}_k^U$ as comparing all $w_i^U \in \widehat{W}_k^U$ with the word $w_{i,j} \in \widehat{W}_p^D$. That is,

$$w_{max}^{U} = arg \max_{w_i^U \in W_k^U} \text{Sim}(w_{i,j}, w_k^U) \tag{22}$$

Assume that w_{max}^{U} are the x-dimension and y-dimension in $\widehat{W}_k^U$. Then, the value of similarity and word $Sim(w_{i,j}, w_{max}^{U})$ will be stored in the x-th and y-th dimension values, respectively. That is,

$$\begin{aligned} \Phi(i,x).sim &= Sim(w_{i,j}, w_{\max}^{U}) \\ \Phi(i,y).word &= Sim\left(w_{i,j}, w_{\max}^{U}\right) \end{aligned} \tag{23}$$

Till now, each document has obtained its document vector and the words $w \in W_k^U$. The next step is to adopt the document vector to find the most similar document in U (D^q) with the document D^q. Let $d(D^i, D^j)$ denote the distance of documents D^i and D^j. The Euclidean distance will be adopted to measure the distance between the two documents. That is,

$$d(D^i, D^j) = \left(\sum_{x=1}^{k} \left[\Phi(i,x) - \Phi(j,x) \right]^2 \right)^{\frac{1}{2}} \tag{24}$$

Let D^{like} be the document that is the most similar with D^q. That is,

$$D^{\text{like}} = \arg \min_{D^i \in U - \{D^q\}} d\left(D^i, D^q\right) \tag{25}$$

As shown in Eq. (25), the document D^{like}, which is the most similar to D^q, can be found according to the Euclidean distance of $\Phi(D^q)$ and $\Phi(D^{\text{like}})$. The next step is to find similar words, sentences as well as paragraphs of D^q and D^{like}. To achieve this, this paper proposes a low comparison-cost and quick mechanism, based on the Joint Coarse and Fine-Grained (*JCF*) policy. The *JCF* policy first compares the sentence similarity, then paragraph similarity and finally document similarity. The document vectors of D^q and D^{like} have depicted their similar words. The first step is to identify similar words in the two documents and then extract the corresponding sentences for verification.

The following gives an example to illustrate the design concept. Assume that the document vectors $\Phi(D^q)$ and $\Phi(D^{\text{like}})$ are.

$$\begin{aligned} \Phi(D^q) &= \left[\left(0.85, \text{good}'\right), \left(0.3, \text{teacher}'\right), \ldots \right] \text{ and} \\ \Phi(D^{\text{like}}) &= \left[\left(0.9, \text{better}'\right), \left(0.15, \}\text{happy}'\right), \ldots \right] \end{aligned} \tag{26}$$

Let W^{sim} be an empty collection · That is,$W^{\text{sim}} = \varnothing$. Assume that the document is a vector of length 1000. The document vectors $\Phi(D^q)$ and $\Phi(D^{\text{like}})$ will be compared one by one to check the similar document features. Let δ_{word} be a threshold value for determining whether or not the two document features are similar. For example, assume δ_{word}. The next step is to verify whether or not the similarity of words $\Phi(D^q).word$ and $\Phi(D^{\text{like}}).word$ is larger than the threshold valueδ_{word}. The following presents a word similarity condition for the i-th words of $\Phi(D^q)$ and $\Phi(D^{\text{like}})$.

Word similarity check condition for i-th words(*i-WSC*)

$$\min(\Phi(D^q, i).\,sim,\ \Phi(D^{\text{like}}, i).\,sim) \geq \delta_{\text{word}} \tag{27}$$

Let Boolean variable μ_i^{word} represent whether or not the i-th words of $\Phi(D^q)$ and $\Phi(D^{\text{like}})$ satisfy the word similarity condition. That is,

$$\mu_i^{\text{word}} = \begin{cases} 1 & if\ (i - WSC)\ is\ satisfied \\ 0 & other\ wise \end{cases} \tag{28}$$

For example, μ_1^{word} is 1. This occurs because the first words of $\Phi(D^q)$ and $\Phi(D^{\text{like}})$ satisfy the 1-WSC condition

$$\begin{aligned} &\min(\Phi(D^q, i).\,\text{sim},\ \Phi(D^{\text{like}}, i).\text{sim}) \\ &= \min(0.85, 0.9) = 0.85 \geq 0.8 \end{aligned} \tag{29}$$

Since the two words 'good' and 'better' are similar, the pair of the two words should be stored in W^{sim}. Then, we have a pair of similar word

$$W^{\text{sim}} = \left\{ \left('\text{good}', '\ \text{better}' \right) \right\} \tag{30}$$

The above-mentioned operations should be performed on the i-th words of $\Phi(D^q)$ and $\Phi(D^{\text{like}})$ for $1 \leq i \leq k$. That is, the following *word similarity check* operations should be performed:

for i in range(1, *k*):

if $\mu_i^{\text{word}} == 1$:

$$W^{\text{sim}} = W^{\text{sim}} \cup (\Phi\ (D^q, i).\ word, \Phi\ (D^{\text{like}}, i).\ word)$$

Till now, all similar words (document features) of the two documents D^q and D^{like} have been stored in the set W^{sim}. The next subsection further checks the similarity of those sentences which contain the word pair in W^{sim}.

4.3 Fine grain: sentence similarity check phase

In the last subsection, all similar words (document features) of the two documents D^q and D^{like} have been stored in a set W^{sim}. This subsection aims to check the similarity of D^q and D^{like} in the sentence level. Let (x, y) be one element of W^{sim}. For each $(x, y) \in W^{\text{sim}}$, the following presents the similarity check at the sentence level. Let S_x^q denote the set of sentences containing the word x in the document D^q. Let S_y^{like} denote the set of sentences containing the word y in the document D^{like}.

Let $V_{S_x^q}^{\text{sentence}}$ denote the set of sentence vectors of the sentence set S_x^q. Let $V_{S_y^{\text{like}}}^{\text{sentence}}$ denote the set of sentence vectors of the sentence set S_y^{like}.

The similar sentences of S_x^q and S_y^{like} will be found based on the calculation of the sentence vector. Let $s = (w_1, w_2, w_3, \ldots, w_r)$ represent a sentence, which consists of r words. The following will introduce the implementation of a function $s2v(s)$ which transfers a sentence s to a sentence vector. Let v_i denote the word vector of w_i. Let V_S represent the sentence vectors $V_S = (v_1, \ldots, v_r)$ of sentence $s = (w_1, w_2, w_3, \ldots, w_r)$. Let

$\overline{v}_s$ denote the average sentence vector which is obtained by the average vector of V_S. The following presents the pseudo-code of function s2v().

Def s2v(s):

Let $s=(w_1, w_2, w_3,\ldots, w_r)$

Transfer each word $w_i \in$ s to v_i

Let $V_S=(v_1,\ldots, v_r)$

$\overline{v}_s=\sum_{i=1}^{r} v_i/r$

return($\overline{v}_s$)

Initially, assume that $V_{S_x^q}^{\text{sentence}}$ and $V_{S_y^{\text{like}}}^{\text{sentence}}$ both are empty sets · That is,

$$V_{S_x^q}^{\text{sentence}} = \emptyset,\ V_{S_y^{\text{like}}}^{\text{sentence}} = \emptyset \tag{31}$$

The next step is to apply function $s2v()$ to transfer s to a sentence vector $\overline{v}_s$ for each sentence $s \in S_x^q$. That is,

$$V_{S_x^q}^{\text{sentence}} = V_{S_x^q}^{\text{sentence}} \cup \overline{v}_s \tag{32}$$

Similarly, the function $s2v()$ is applied to transfer $\hat{s}$ to a sentence vector $\overline{v}_{\hat{s}}$ for each sentence $\hat{s} \in S_y^{\text{like}}$. Then the set $V_{S_y^{\text{like}}}^{\text{sentence}}$ is updated accordingly.

$$V_{S_y^{\text{like}}}^{\text{sentence}} = V_{S_y^{\text{like}}}^{\text{sentence}} \cup \overline{v}_{\hat{s}} \tag{33}$$

Let v_i^q belongs to $V_{S_x^q}^{\text{sentence}}$ and v_j^{like} belongs to $V_{S_y^{\text{like}}}^{\text{sentence}}$. Let $V_{x,y}^{\text{sim}}$ denote an empty set. All possible combinational pairs $(v_i^q, v_j^{\text{like}})$ for $v_i^q \epsilon V_{S_x^q}^{\text{sentence}}$ and $v_j^{\text{like}} \in V_{S_y^{\text{like}}}^{\text{sentence}}$ will be checked the similarity of v_i^q and v_j^{like} by applying the cos-similarity $cos\left(v_i^q, v_j^{\text{like}}\right)$.

The following presents the *SSC* condition which aims to find similar sentences in documents D^q and D^{like}.

Sentence similarity check (*SSC*) *condition*

$$V_{x,y}^{\text{sim}} = \left\{ (v_i^q, v_j^{\text{like}}) | \cos\left(v_i^q, v_j^{\text{like}}\right) \geq \delta_{sentence}, for\ v_i^q \epsilon V_{S_x^q}^{\text{sentence}}, v_j^{\text{like}} \epsilon V_{S_y^{\text{like}}}^{\text{sentence}} \right\} \tag{34}$$

Let $(s_i^q,\ s_j^{\text{like}})$ be the sentence pair of sentence vector pair $(v_i^q, v_j^{\text{like}})$, for all $(v_i^q, v_j^{\text{like}}) \epsilon V_{x,y}^{\text{sim}}$. Let S^{sim} denote the set of all pairs $(s_i^q, s_j^{\text{like}})$. we have

$$S^{\text{sim}} = \left\{ (s_i^q,\ s_j^{\text{like}})\ \forall\ (v_i^q,\ v_j^{\text{like}}) \epsilon V_{x,y}^{\text{sim}} \right\} \tag{35}$$

Till now, all similar sentences of the two documents D^q and D^{like} have been stored in a set S^{sim}. The next subsection further checks the similarity of those paragraphs which contain the sentence pair in S^{sim}.

4.4 Fine grain: paragraph similarity check phase

In the last subsection, all similar sentences of the two documents D^q and D^{like} have been stored in a set S^{sim}. This subsection aims to check the similarity of D^q and D^{like} in the paragraph level for those similar sentences. Let $(s_i^q, s_j^{\text{like}})$ be one element of S^{sim}. For each $(s_i^q, s_j^{\text{like}}) \in S^{\text{sim}}$, the following presents the similarity check at the paragraph level. Let $P_{s_i}^q$ denote the set of paragraphs containing the sentence s_i^q. Let $P_{s_j}^{\text{like}}$ denote the set of paragraphs containing s_j^{like}. Let $V_{P_{s_i}^q}^{\text{paragraph}}$ denote the set of paragraph vectors of the paragraph set S_x^q. Let $V_{P_y^{\text{like}}}^{\text{paragraph}}$ denote the set of sentence vectors of the paragraph set P_y^{like}.

The similar check of paragraphs P_x^q and P_y^{like} will be found based on the calculation of the sentence vector. The similar check of paragraph sets $P_{s_i}^q$ and $P_{s_j}^{\text{like}}$ will be done based on the calculation of the paragraph vector. Let $p = (s_1, s_2, s_3, \ldots, s_r)$ represent paragraph, which consists of r sentences. Let $p2v(p)$ be a function that converts a paragraph s to a paragraph average vector.

The following will introduce the implementation of a function $p2v(p)$ which transfers a paragraph p to a paragraph vector. Let v_i denote the sentences vector of s_i. Let V_p represent the paragraphs' vectors $V_p = (v_1, \ldots, v_r)$ of paragraph $p = (s_1, s_2, s_3, \ldots, s_r)$. Let $\overline{v}_p$ denote the average paragraph vector which is obtained by the average vector of V_p. The following presents the pseudo-code of function $p2v()$.

Def p2v(p):

$p = (s_1, s_2, s_3, \ldots, s_r)$

Transfer each sentence $s_i \in p$ to v_i

Let $V_p = (v_1, \ldots, v_r)$

$V_p = (v_1 = s2v(s_1), \ldots, v_r = s2v(s_r))$

$\overline{v}_p = \sum_{i=1}^{r} v_i / r$

return$(\overline{v}_p)$

Initially, assume that $V_{P_i^q}^{\text{paragraph}}$ and $V_{P_j^{\text{like}}}^{\text{paragraph}}$ both are empty sets . That is,

$V_{P_i^q}^{\text{paragraph}} = \varnothing, V_{P_j^{\text{like}}}^{\text{paragraph}} = \varnothing$

The next step is to apply function $p2v()$ to transfer p to a paragraph vector $\overline{v}_p$ for each paragraph $p \in P_{s_i}^q$. That is,

for each p in $P_{s_i}^q$:

$\overline{v}_p = p2v(p)$

$V_{P_{s_i}^q}^{\text{paragraph}} = V_{P_{s_i}^q}^{\text{paragraph}} \cup \overline{v}_p$

Similarly, the function $p2v()$ is applied to transfer $\widehat{p}$ to a paragraph vector $\overline{v}_{\widehat{p}}$ for each sentence $\widehat{p} \in P^{\text{like}}_{s_j}$. Then, the set $V^{\text{paragraph}}_{P^{\text{like}}_{s_j}}$ is updated accordingly

for each $\widehat{p}$ in $P^{\text{like}}_{s_j}$:

$$\overline{v}_{\widehat{p}} = p2v(\widehat{p})$$

$$V^{\text{paragraph}}_{P^{\text{like}}_{s_j}} = V^{\text{paragraph}}_{P^{\text{like}}_{s_j}} \cup \overline{v}_{\widehat{p}}$$

Let v^q_a belong to $V^{\text{paragraph}}_{P^q_{s_i}}$ and v^{like}_b belong to $V^{\text{paragraph}}_{P^{\text{like}}_{s_j}}$. Let $V^{\text{sim}}_{a,b}$ denote an empty set. All possible combinational pairs $(v^q_a, v^{\text{like}}_b)$ for $v^q_a \in V^{\text{paragraph}}_{P^q_{s_i}}$ and $v^{\text{like}}_z \in V^{\text{paragraph}}_{P^{\text{like}}_{s_j}}$ will be checked for the similarity of v^q_a and v^{like}_b by applying the cos-similarity $cos\left(v^q_a, v^{\text{like}}_b\right)$.

The following presents the *PSC* condition which aims to find a similar paragraph of documents D^q and D^{like}.

Let $P^{\text{sim}}=\{(p^q_a, p^{\text{like}}_b),\ldots\}$, where $(p^q_a, p^{\text{like}}_b)$ is a paragraph pair of vector pair $(p^q_a, p^{\text{like}}_b)$.

Paragraph similarity check (PSC) condition

$$V^{\text{sim}}_{t,z} = \left\{ \left(v^q_a, v^{\text{like}}_b\right) \middle| \cos\left(v^q_a, v^{\text{like}}_b\right) \geq \delta_{\text{paragraph}}, \text{for each } v^q_a \epsilon V^{\text{paragraph}}_{P^q_{s_i}}, v^{\text{like}}_b \epsilon V^{\text{paragraph}}_{P^{\text{like}}_{s_j}} \right\} \tag{36}$$

Let $(p^q_a, p^{\text{like}}_b)$ be the paragraph pair of paragraph vector pair $(v^q_a, v^{\text{like}}_b)$, for all $(v^q_a, v^{\text{like}}_b)\epsilon V^{\text{sim}}_{a,b}$. Let P^{sim} denote the set of all pairs $(p^q_t, p^{\text{like}}_z)$. we have

$$P^{\text{sim}} = \left\{ (p^q_a, p^{\text{like}}_b),\ \forall\, (v^q_a, v^{\text{like}}_b)\epsilon V^{\text{sim}}_{a,b} \right\} \tag{37}$$

Till now, all similar paragraphs of the two documents D^q and D^{like} have been stored in a set P^{sim}. The next subsection further checks the similarity of those documents which contain the paragraph pair in P^{sim}.

4.5 Fine grain: document similarity check phase

After finding the most similar paragraph, the next step is the word-to-word comparison of the entire document. To present the comparison method, some notations are defined. Let $\lambda^{\text{paragraph}}_{q,i}$ denote the number of paragraphs in the document D^q. Let $\lambda^{\text{sentence}}_{q,i,j}$ denote the number of sentences in the paragraph $p_{q,i}$ of document D^q. Let word $\lambda^{\text{word}}_{q,i,j,k}$ denote the number of words in the sentence $s_{q,i,j}$ of paragraph $p_{q,i}$ of document D^q.

The next step is to calculate the document average vector in each stage (paragraph, sentence and word). Let $\overline{V}^q_{S_{q,i,j}}$ denote the average sentence vector of the sentence $s_{q,i,j}$ in paragraph $p_{q,i}$ of D^q. The value of $\overline{V}^q_{s_{q,i,j}}$ can be derived by

$$\overline{V}^{q}_{s_{q,i,j}} = \frac{\sum_{w_{q,i,j,k} \in s_{q,i,j}} \overline{V}_{w_{q,i,j,k}}}{\lambda^{\text{word}}_{q,i,j}} \tag{38}$$

Similarly, let $\overline{V}^{q}_{p_{q,i}}$ denote the average paragraph vector, which can be derived by sentence vectors as follows.

$$\overline{V}^{q}_{p_{q,i}} = \frac{\sum_{s_{q,i,j} \in p_{q,i}} \overline{V}_{s_{q,i,j}}}{\lambda^{\text{sentence}}_{q,i}} \tag{39}$$

Let $\overline{V}^{q}_{D}$ denote the average document vector of D^q, which is derived from the paragraph vectors of D^q. That is,

$$\overline{V}^{q}_{D} = \frac{\sum_{p_{q,i} \in D^q} \overline{V}_{p_{q,i}}}{\lambda^{\text{paragraph}}_{q}} \tag{40}$$

After calculating the average vectors of all paragraphs and sentences, put these obtained vectors into the paragraph vector set and sentence vector set, respectively. Let V^{q}_{D} be the set of paragraph vectors. That is,

$$V^{q}_{D} = \{\overline{V}^{q}_{p_{q,1}}, \overline{V}^{q}_{p_{q,2}}, \ldots, \overline{V}^{q}_{p_{\lambda^{\text{paragraph}}_{q}}}\} \tag{41}$$

Let $V^{q}_{p_i}$ be the set of sentences vectors in paragraph $p_{q,i}$. That is,

$$V^{q}_{p_i} = \{\overline{V}^{q}_{s_{q,i,1}}, \overline{V}^{q}_{s_{q,i,2}}, \ldots, \overline{V}^{q}_{s_{q,i,\lambda^{\text{sentence}}_{q,i}}}\} \tag{42}$$

Let $V^{q}_{s_{i,j}}$ be the set of words vectors in the sentence $s_{q,i,j}$ in paragraph $p_{q,i}$,

$$V^{q}_{s_{i,j}} = \{\overline{V}^{q}_{W_{q,i,j,1}}, \overline{V}^{q}_{W_{q,i,j,2}}, \ldots, \overline{V}^{q}_{W_{q,i,j,\lambda^{\text{word}}_{q,i,j}}}\} \tag{43}$$

Similarly, the next step is to calculate the vector of the document D^{like}. Let $\lambda^{\text{paragraph}}_{\text{like}}$ denote the number of paragraphs in the document D^{like}. Let $\lambda^{\text{sentence}}_{like,m}$ denote the number of sentences in the paragraph $p_{like,m}$ of document D^{like}. Let word $\lambda^{\text{word}}_{like,m,n}$ denote the number of words in the sentence $s_{like,m,n}$ of paragraph $p_{like,m}$ of document D^{like}.

The next step is to calculate the document average vector in each stage (paragraph, sentence and word). Let $\overline{V}^{\text{like}}_{s_{like,m,n}}$ denote the average sentence vector of the sentence $s_{like,m,n}$ in paragraph $p_{like,m}$ of D^{like}. The value of $\overline{V}^{\text{like}}_{s_{like,m,n}}$ can be derived by

$$\overline{V}^{\text{like}}_{s_{like,m,n}} = \frac{\sum_{w_{like,m,n,o} \in s_{m,n}} \overline{V}_{w_{like,m,n,o}}}{\lambda^{\text{word}}_{like,m,n}} \tag{44}$$

Algorithm: ***Word_Output***

Input: the two documents D^q and D^{like}

Output: the corresponding words $w_{q,i,j,k}$ and $w_{like,m,n,o}$

1. *For each* $V_{p_i}^q \in V_D^q, V_{p_m}^{like} \in V_D^{like}$
2. *If sim*$(V_{p_i}^q, V_{p_m}^{like}) \geq \delta_{paragraph}$
3. *For each* $V_{s_{i,j}}^q \in V_{p_i}^q$, $V_{s_{m,n}}^{like} \in V_{p_m}^{like}$
4. *If sim*$(V_{s_{i,j}}^q, V_{s_{m,n}}^{like}) \geq \delta_{sentence}$
5. *For each* $w_{q,i,j,k} \in V_{s_{i,j}}^q$, $w_{like,m,n,o} \in V_{s_{m,n}}^{like}$
6. *If sim*$(w_{q,i,j,k}, w_{like,m,n,o}) \geq \delta_{word}$
7. *Output*$(w_{q,i,j,k}, w_{like,m,n,o})$

Fig. 2 Word_Output algorithm

Similarly, let $\overline{V}_{p_{like,m}}^{\text{like}}$ denote the average paragraph vector, which can be derived by sentence vectors as follows.

$$\overline{V}_{p_{like,m}}^{\text{like}} = \frac{\sum_{s_{like,m,n} \in p_{like,m}} \overline{V}_{s_{like,m,n}}}{\lambda_{like,m}^{\text{sentence}}} \tag{45}$$

Let $\overline{V}_D^{\text{like}}$ denote the average document vector of D^{like}, which is derived from the paragraph vectors of D^{like}. That is,

$$\overline{V}_D^{\text{like}} = \frac{\sum_{p_{like,m} \in D^{\text{like}}} \overline{V}_{p_{like,m}}}{\lambda_{\text{like}}^{\text{paragraph}}} \tag{46}$$

Let V_D^{like} be the set of paragraph vectors. That is,

$$V_D^{\text{like}} = \{\overline{V}_{p_{like,1}}^{\text{like}}, \overline{V}_{p_{like,2}}^{\text{like}}, \ldots, \overline{V}_{p_{like,\lambda_{\text{like}}^{\text{paragraph}}}}^{\text{like}}\} \tag{47}$$

Let $V_{p_m}^{\text{like}}$ be the set of sentences vectors in paragraph $p_{like,m}$. That is,

$$V_{p_m}^{\text{like}} = \{\overline{V}_{s_{like,m,1}}^{\text{like}}, \overline{V}_{s_{like,m,2}}^{\text{like}}, \ldots, \overline{V}_{s_{like,m,\lambda_{like,m}^{\text{sentence}}}}^{\text{like}}\} \tag{48}$$

Let $V_{s_{m,n}}^{\text{like}}$ be the set of words vectors in the sentence $s_{like,m,n}$ in paragraph $p_{like,m}$,

$$V_{s_{m,n}}^{\text{like}} = \{\overline{V}_{w_{like,m,n,1}}^{\text{like}}, \overline{V}_{w_{like,m,n,2}}^{\text{like}}, \ldots, \overline{V}_{w_{like,m,n,\lambda_{like,m,n}^{\text{word}}}}^{\text{like}}\} \tag{49}$$

The next step aims to find the most similar words between the two documents D^q and D^{like}. Figure 2. gives the *Word_Output* algorithm which aims to find all similar words in the two documents. First, similar paragraphs should be found. It can be achieved by comparing the paragraph vector $V_{p_i}^q \in V_D^q$ and paragraph vector $V_{p_m}^{\text{like}} \in V_D^{\text{like}}$. In case, the similarity of $V_{p_i}^q$ and $V_{p_m}^{\text{like}}$ exceeds the predefined threshold value $\delta_{\text{paragraph}}$, the sentences in the two corresponding paragraphs should be further checked. Steps 1 and 2 examine the similarity of paragraphs p_m and p_i. If it is the case, the next step is to find similar sentences in p_m and p_i. Steps 3 and 4 further compare the similarity of sentence vectors $V_{S_{i,j}}^q \in V_{p_i}^q$ and $V_{S_{m,n}}^{\text{like}} \in V_{p_m}^{\text{like}}$. In case the similarity of $V_{S_{i,j}}^q$ and $V_{S_{i,j}}^q$ exceeds the predefined threshold value $V_{S_{i,j}}^q$, the words in the two sentences should be further compared. Steps 5 and 6 aim to check the similarity in word level for documents D^q and D^{like}.

In case the word vectors V_W^q and V_W^{like} is larger than the threshold value δ_{word}, the corresponding words $w_{q,i,j,k}$ and $w_{like,m,n,o}$ will be outputted. The proposed *JCF* adopts coarse-grained and then fine-grained comparison. It aims to speed up the similarity comparison. Therefore, the *JCF* applies the Bag of Words scheme to initially establish a set of common features of all documents and then transforms each document into a document vector. Then, the most similar documents can be found as the candidate document.

After that, the fine-grained comparison is applied to the candidate documents which are found in the previous step. the candidate document is compared with the target document in a manner of fine grain, which compares the similarity in sentence level and then paragraph level. In the comparison of sentence level, the proposed *JCF* only compares those sentences which contain similar words found in the coarse-grained comparison. After that, in the comparison of paragraph level, the *JCF* only compares the paragraphs which contain a certain number of similar sentences. The selective comparison not only speeds up the process but also facilitates the extraction of semantic similarity at the sentence and paragraph levels.

4.6 Similarity restoration

It is impossible to obtain the similarity between two documents by comparing them word by word since it is time-consuming. This study extracts features with different levels (document level, sentence level and paragraph level) to present the document, sentence and paragraph and quickly obtain document similarity based on these features. However, the similarity estimation by features such as TF–IDF keywords, sentence vectors or even paragraph vectors may have a slight error with the actual similarity. Therefore, a similarity restoration mechanism is required to achieve the truest similarity. This can reduce the estimation error on similarity, which plays an important role in our design. First of all, it is necessary to generate document samples. The main purpose is to calculate the estimation error of similarity between the

Table 2 Simulation settings

Keyword	50/100/300/500
Threshold	0.3/0.5/0.7/0.9
Data source	Wiki corpus provided by Wikipedia
Number of dataset documents	410,000
Epoch	10
Train data/Test data	410,000/1,000

JCF method and the actual similarity when the similarity of the article is known. A sample is generated by removing a certain percentage of a document and keeping the rest. For example, 20% of document A is removed and only 80% of document A is reserved. Then, the removed 20% should be replaced by the contents selected from other documents.

After obtaining a new document, say document B, it is known that the actual similarity of documents A and B is 80%. Then, the proposed *JCF* can be applied to estimate the similarity of the two documents A and B. Therefore, the estimation error of *JCF* can be measured. For example, the *JCF* adopts the extracted features and detects the 70% similarity between documents A and B. Then, the actual similarity of the two documents can be restored from 70 to 70%* (80%/70%) = 80%. After calculating the similarity of articles with different proportions many times, the error results are counted to obtain the average function σ=80%/70%. Through the above method, after sending a new document into *JCF* to calculate the similarity value, the similarity restoration function σ will be applied to obtain the actual similarity.

5 Performance evaluation

This section mainly compares the performance of the proposed algorithm and the Bert-based and Word2Vec-based algorithms in Plagiarism Detection. The experimental setup is described below. The experimental setup is described below. As shown in Table 2, the performance results highly depend on the similarity thresholds and the number of keywords. Therefore, the similarity thresholds would be varied ranging from 0.3 to 0.9 while the number of keywords is varied ranging from 50 to 500.

Table 3 Threshold value set of sample A accuracy

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.71	0.79	0.76	0.89
Threshold 0.5	0.72	0.86	0.8	0.94
Threshold 0.7	0.67	0.77	0.73	0.85
Threshold 0.9	0.61	0.75	0.7	0.84

Table 4 Threshold value set of sample A precision

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.63	0.704	0.68	0.815
Threshold 0.5	0.675	0.805	0.786	0.916
Threshold 0.7	0.72	0.85	0.83	0.963
Threshold 0.9	0.72	0.85	0.83	0.963

Table 5 Threshold value set of sample A recall

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.876	0.95	0.916	0.99
Threshold 0.5	0.91	0.97	0.95	1
Threshold 0.7	0.83	0.89	0.86	0.916
Threshold 0.9	0.76	0.83	0.8	0.853

Table 6 Threshold value set of sample A F1-score

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.704	0.812	0.791	0.879
Threshold 0.5	0.792	0.879	0.858	0.946
Threshold 0.7	0.773	0.861	0.84	0.907
Threshold 0.9	0.723	0.811	0.79	0.877

The software platform of the experiment in this research is Windows 10 operating system, and the development environment is Python 3.7.13. The CPU Intel core i9-10900 k is used for model training and experiments. The metrics for comparing the proposed algorithm and the existing Bert and Word2Vec are the accuracy, precision, recall, and F1-score of the three algorithms. MATLAB is used as the image display of data.

The experiments in this research will be analyzed based on two training sets *A* and *B*. The samples in *A* and *B* are all self-generated documents. To match different degrees of plagiarism, the similarity range of the generated articles is set between 40 and 80%. The documents in training set *A* are all generated after partial replacement with words unrelated to the original content, while the documents in training set *B* are generated by replacing part of the article content with similar word vectors which are generated based on the Word2Vec mechanism. The difference between sets *A* and *B* aims to measure whether or not the plagiarism detection algorithms can detect the similarity at the semantic level. In case the similarity comparison only

Table 7 Value set of sample A precision

			Threshold			
			0.3	0.5	0.7	0.9
JCF	Keyword	50	0.77	0.85	0.95	0.95
		100	0.79	0.87	0.96	0.95
		300	0.83	0.90	0.96	0.97
		500	0.83	0.92	0.96	0.97
Study [14] based on Word2Vec	Keyword	50	0.74	0.84	0.94	0.94
		100	0.74	0.84	0.94	0.94
		300	0.74	0.84	0.94	0.94
		500	0.74	0.84	0.94	0.94
Study [13] based on Bert	Keyword	50	0.69	0.83	0.94	0.92
		100	0.69	0.83	0.94	0.92
		300	0.69	0.83	0.94	0.92
		500	0.69	0.83	0.94	0.92

Table 8 Value set of sample A recall

			Threshold			
			0.3	0.5	0.7	0.9
JCF	Keyword	50	0.42	0.56	0.97	0.96
		100	0.42	0.60	0.97	0.96
		300	0.44	0.70	0.92	0.96
		500	0.44	0.72	0.91	0.96
Study [14] based on Word2Vec	Keyword	50	0.36	0.54	0.90	0.95
		100	0.36	0.54	0.90	0.95
		300	0.36	0.54	0.90	0.95
		500	0.36	0.54	0.90	0.95
Study [13] based on Bert	Keyword	50	0.31	0.43	0.88	0.95
		100	0.31	0.43	0.88	0.95
		300	0.31	0.43	0.88	0.95
		500	0.31	0.43	0.88	0.95

performs on the word-to-word comparison, it is difficult to detect the plagiarism that existed between two different words where the distance of their word vectors is small.

Table 9 Value set of sample A F1-score

			Threshold			
			0.3	0.5	0.7	0.9
JCF	Keyword	50	0.52	0.72	0.89	0.87
		100	0.57	0.74	0.91	0.88
		300	0.61	0.82	0.92	0.90
		500	0.61	0.82	0.92	0.90
Study [14] based on Word2Vec	Keyword	50	0.48	0.66	0.87	0.85
		100	0.48	0.66	0.87	0.85
		300	0.48	0.66	0.87	0.85
		500	0.48	0.66	0.87	0.85
Study [13] based on Bert	Keyword	50	0.47	0.65	0.85	0.82
		100	0.47	0.65	0.85	0.82
		300	0.47	0.65	0.85	0.82
		500	0.47	0.65	0.85	0.82

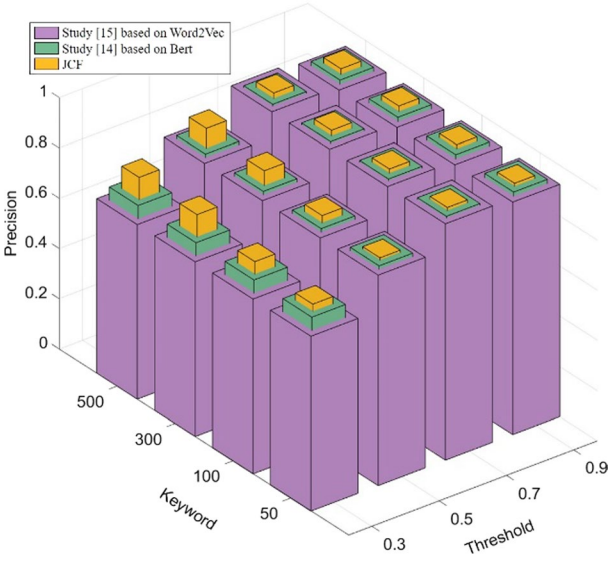

Fig. 3 Precision comparison results of sample A

5.1 Sample A

The following are the experimental results of sample A. The documents in training set A are all generated after partial replacement with words unrelated to the original

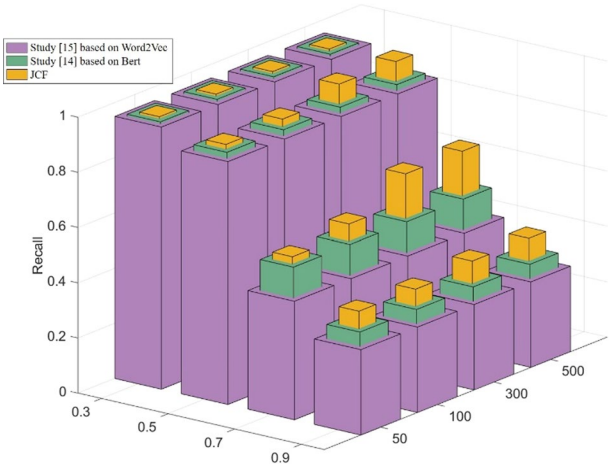

Fig. 4 Recall comparison results of sample A

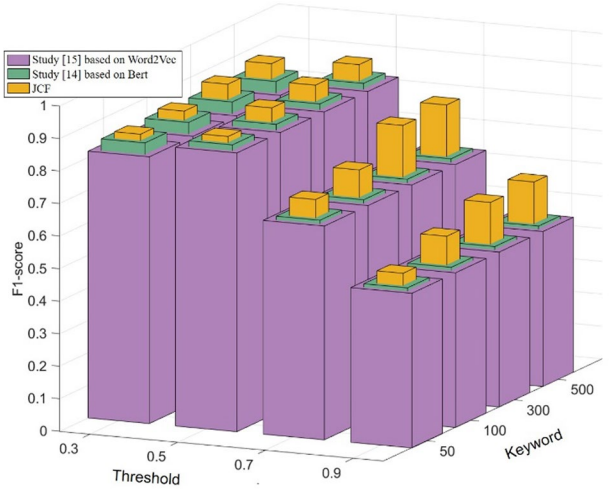

Fig. 5 F1-score comparison results of sample A

content.

Tables 3, 4, 5, and 6 mainly compare the *JCF* Algorithms threshold settings at different stages in terms of accuracy, prediction, recall and F1-score. The main of the experiment is to obtain the best combination of thresholds keywords, sentences and paragraphs. The keywords thresholds are 0.2 and 0.5, while the sentences threshold are 0.2 and 0.5. The experimental results show the common trend that the performance similarity threshold value of different comparison stages is 0.2, the accuracy rate is the lowest, the similarity threshold value of different comparison stages is 0.5, and the accuracy rate is the highest. It can be seen from the above, when the

Table 10 Threshold F1-score set of sample B

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.667	0.75	0.71	0.855
Threshold 0.5	0.724	0.835	0.795	0.89
Threshold 0.7	0.62	0.73	0.697	0.81
Threshold 0.9	0.58	0.71	0.667	0.801

threshold combination of keywords and sentences is both 0.2, the overall accuracy rate is the lowest. The best results are obtained when the thresholds for keywords, sentences and paragraphs are all set to 0.5. This is because when the threshold is set too high, the number of detected documents will be too small. On the contrary, if the threshold is set too low, most of the information in the documents will be screened, and the purpose of setting the threshold will be lost.

The previous paragraph explained the impact of the number of keywords in each article on the comparison results. However, the factors that affect the comparison results also include the similarity threshold. If the threshold is set too high, the system will detect too few articles. If the threshold is set too low, most of the information in the article will be included in the screening, and the purpose of setting the threshold will be lost.

Related studies [13, 14] based on Bert and Word2Vec also use the same evaluation method. For example, these studies [13, 14] based on *Bert* and Word2Vec first convert the sentences in the article into sentence vectors and then sum up the sentence vectors to obtain the average value as the article similarity. In addition to applying the concept of similarity comparison in this study, a similarity restoration algorithm is added to restore each stage in the comparison process to avoid

Table 11 Threshold accuracy set of Sample B

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.59	0.665	0.63	0.775
Threshold 0.5	0.655	0.819	0.756	0.876
Threshold 0.7	0.68	0.81	0.794	0.923
Threshold 0.9	0.68	0.81	0.796	0.923

Table 12 Threshold accuracy set of Sample B

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.816	0.889	0.865	0.94
Threshold 0.5	0.86	0.912	0.894	0.95
Threshold 0.7	0.78	0.842	0.81	0.857
Threshold 0.9	0.705	0.766	0.751	0.794

Table 13 Threshold recall set of Sample B

	Keyword 0.2 sentence 0.2	Keyword 0.2 sentence 0.5	Keyword 0.5 sentence 0.2	Keyword 0.5 sentence 0.5
Threshold 0.3	0.654	0.762	0.745	0.83
Threshold 0.5	0.742	0.827	0.8	0.905
Threshold 0.7	0.723	0.81	0.794	0.857
Threshold 0.9	0.67	0.766	0.735	0.825

errors caused by taking the average value of vectors as the similarity of articles. This makes the final similarity result more accurate.

Tables 7, 8 and 9, and Figs. 3, 4 and 5 mainly compare the performances of the proposed *JCF* and the existing studies [13, 14] based on Bert and Word2Vec algorithms in terms of precision, recall and F1-score. The number of keywords varies ranging from 50 to 300, while the number of thresholds varies ranging from 0.3 to 0.9. The experimental results show the common trend that the performances of the three *JCF* mechanisms are increased with the number of keywords, in terms of prediction, recall and F1-score. This is because the number of features presenting the document is increased with the number of keywords. In addition, as shown in Figs. 4 and 5, the *JCF* mechanism has the best recall and F1-score when the value of the threshold is 0.5. The main reason is that when the threshold value exceeds 0.7, the similarity threshold is too high to find similar sentences, paragraphs and documents. On the contrary, if the threshold value is too smaller, too many sentences, paragraphs and documents are classified to be similar, but they are not similar. In comparison, the proposed *JCF* algorithm is superior to the studies [13, 14] based on Bert and Word2Vec algorithms in all cases. This occurs because the proposed *JCF* adopts the similarity restoration to calculate the similarity in each comparison stage. On the contrary, the studies [13, 14] based on Bert and Word2Vec algorithms only calculate similarity using features. As a result, the proposed *JCF* outperforms the studies [13, 14] based on Bert and Word2Vec Algorithms in terms of precision, recall as well as F1-score.

5.2 Sample B

The following depicts the experimental results of sample B. Sample B is generated by using similar words to replace part of the document content. The removed and replaced words are similar in word vectors. Therefore, it is more challenging to identify the content similarity of the documents, as compared with sample A.

Table 14 Operation time required for sample B

	Study [14] based on Word2Vec	Study [13] based on Bert	*JCF*
10,000	74.4	68.2	60.8
15,000	88.5	82.7	77.5
20,000	118.7	110.5	105.8
25,000	136.2	132	128.8

Table 15 Value set of sample B precision

			threshold			
			0.3	0.5	0.7	0.9
JCF	Keyword	50	0.74	0.84	0.93	0.94
		100	0.76	0.84	0.94	0.95
		300	0.83	0.90	0.96	0.96
		500	0.83	0.90	0.96	0.96
Study [14] based on Word2Vec	Keyword	50	0.69	0.83	0.95	0.94
		100	0.69	0.83	0.95	0.94
		300	0.69	0.83	0.95	0.94
		500	0.69	0.83	0.95	0.94
Study [13] based on Bert	Keyword	50	0.67	0.80	0.95	0.94
		100	0.67	0.80	0.95	0.94
		300	0.67	0.80	0.95	0.94
		500	0.67	0.80	0.95	0.94

Table 16 Value set of sample B Recall

			Threshold			
			0.3	0.5	0.7	0.9
JCF	Keyword	50	0.35	0.56	0.88	0.96
		100	0.40	0.60	0.92	0.96
		300	0.44	0.70	0.97	0.96
		500	0.44	0.70	0.97	0.96
Word2Vec Study [14] based on	Keyword	50	0.33	0.49	0.87	0.95
		100	0.33	0.49	0.87	0.95
		300	0.33	0.49	0.87	0.95
		500	0.33	0.49	0.87	0.95
Study [13] based on Bert	Keyword	50	0.31	0.43	0.86	0.95
		100	0.31	0.43	0.86	0.95
		300	0.31	0.43	0.86	0.95
		500	0.31	0.43	0.95	0.95

Tables 10, 11, 12 and 13 mainly compare the performance of the *JCF* Algorithms in terms of accuracy, prediction, recall and F1-score, respectively, under different threshold settings. The keywords thresholds are set at 0.2 and 0.5, while the sentence thresholds are set at 0.2 and 0.5. The experimental results show the common trend that the lowest and highest accuracies have occurred when the threshold values are 0.2 and 0.5, respectively. The recognition difficulty of the B samples is relatively high. When the thresholds of keywords and sentences are both 0.2, the overall

Table 17 Value set of sample B F1-score

			Threshold			
			0.3	0.5	0.7	0.9
JCF	Keyword	50	0.52	0.72	0.89	0.85
		100	0.57	0.74	0.91	0.88
		300	0.61	0.82	0.92	0.90
		500	0.61	0.82	0.92	0.91
Study [14] based on Word2Vec	Keyword	50	0.49	0.65	0.85	0.82
		100	0.49	0.65	0.85	0.82
		300	0.49	0.65	0.85	0.82
		500	0.49	0.65	0.85	0.82
Study [13] based on Bert	Keyword	50	0.47	0.60	0.84	0.80
		100	0.47	0.60	0.84	0.80
		300	0.47	0.60	0.84	0.80
		500	0.47	0.60	0.84	0.80

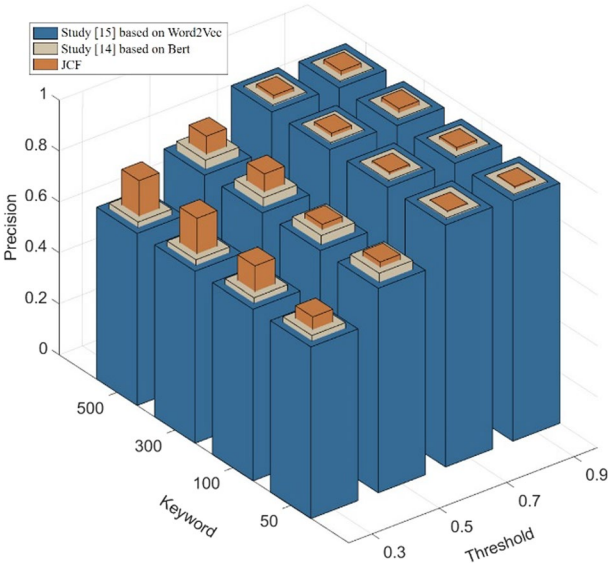

Fig. 6 Precision comparison result of sample B

accuracy is the lowest. The best results are obtained when the thresholds for keywords, sentences and paragraphs are all set to 0.5. This is because when the threshold is set at a large value, the number of detected documents will be small. On the contrary, if the threshold is set at a small value, most of the information in the documents will be screened.

Table 14 mainly compares the performances of the proposed *JCF* and the existing studies [13, 14] based on Bert and Word2Vec Algorithms in terms of CPU running

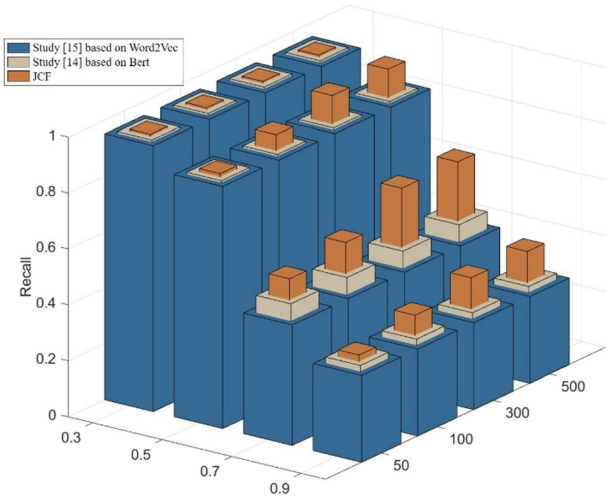

Fig. 7 Recall comparison result of sample B

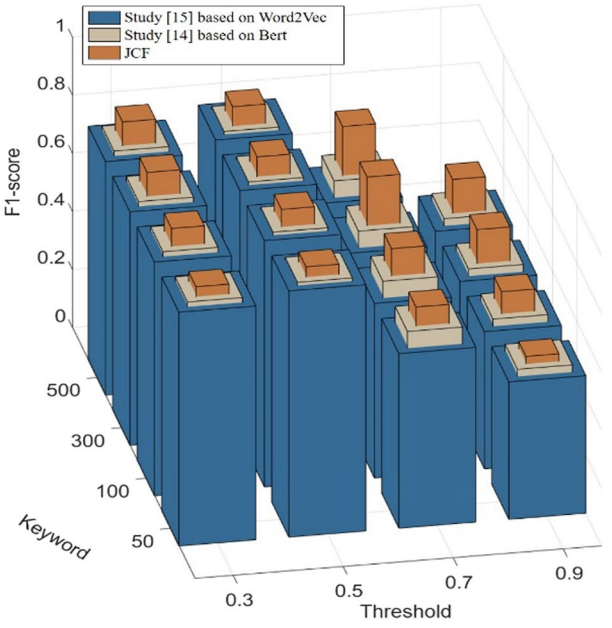

Fig. 8 F1-score comparison result of sample B

time. The document length is varied ranging from 10,000 to 25,000. The experimental results show the common trend that the running times of the three mechanisms are increased with the document length. In comparison, the *JCF* method is significantly faster than other methods. The main reasons are illustrated below. The *JCF* consists

of paragraph, sentence and word stages. In each stage, only the important information of the article is considered for similarity comparison. Therefore, the whole comparison process can be completed in a shorter time. On the contrary, the studies [13, 14] based on Bert and Word2Vec first got the sentence vectors; then, the sentence similarity of the two articles is compared one by one. In general, the proposed *JCF* outperforms the studies [13, 14] based on Bert and Word2Vec algorithms.

Tables 15, 16 and 17, and Figs. 6, 7 and 8 mainly compare the performances of the proposed *JCF* and the existing algorithms in terms of precision, recall and F1-score. The number of keywords varies ranging from 50 to 300, while the number of thresholds varies ranging from 0.3 to 0.9. The experimental results show the common trend that the performances of the three *JCF* mechanisms are increased with the number of keywords, in terms of prediction, recall and F1-score. This is because the number of features presenting the document is increased with the number of keywords. In addition, as shown in Figs. 7 and 8, the *JCF* mechanism has the best recall and F1-score when the value of the threshold is 0.5. The main reason is that the detailed comparison is carried out through many comparison stages when the threshold is set at 0.5. In addition, the proposed *JCF* can recognize similar words and treat them as similar words. On the contrary, the studies [13, 14] based on Bert and Word2Vec were unable to detect similar words to replace, so the training results are worse than *JCF*. As a result, the proposed *JCF* algorithm is superior to the studies [13, 14] based on Bert and Word2Vec algorithms in all cases.

In our study, the method (*JCF*) detected 3,129 similar documents out of a total of 10,000 compared documents, assuming a similarity threshold of over 70%. Studies [13, 14] which were developed based on BERT and Word2Vec found 2985 and 3025 similar documents, respectively. The results of our Z-test achieve a P-value of 0.0027, which is less than 0.05, indicating statistical significance.

The document similarity comparison process involves several phases with varying time complexities. In the Similar Document Identification Phase, the key operation is the TF–IDF. The time complexity is $O(n^2)+O(n^{2*}m)$, where n and m denote the maximal number of words in each document and the number of documents, respectively. Next, in the Word Similarity Check Phase, we assume that the length of the Bag of Words is k. Each document has a vector with length k. Then, the m documents will be sorted to find out the maximum one. The time complexity for sorting is bounded by O(m log m). In the third phase, the Sentence Similarity Check Phase, the comparison of sentence similarity requires $O(k)$ since the maximal number of similar words is k. In the worst case, there are k paragraphs needed to be further compared in paragraph-level comparison. Therefore, the complexity of paragraph-level comparison is O (k). Finally, in the Document Similarity Check Phase, a word-by-word comparison was performed. This required time complexity O (n^2). Therefore, the overall time complexity is $O(n^{2*}m)$.

Finally, the proposed mechanism does not compare a document with all documents in a word-by-word manner since it is inefficient. In fact, this study employed a cost-effective approach. Initially, the TF–IDF scheme is applied to build a bag of words as the representation of the common features of all documents. Then, the plagiarism comparison is carried out in a coarse-grained manner. This can speed up the similarity comparison for the target document. Next, the most similar documents are compared with the target document in detail based on a fine-grained approach.

Therefore, the sentence similarity and document similarity of very few documents will be compared with the target document in detail. As a result, the proposed approach requires low comparison costs.

6 Conclusion

This paper proposes a *JCF* algorithm, aiming at proposing a document similarity comparison mechanism. The *JCF* can reduce the time required for comparing documents. Initially, the TF–IDF scheme is applied to construct a bag-of-words representation that captures the common features of all documents. Plagiarism comparisons are then conducted at a coarse-grained level, which accelerates the similarity comparison of the target documents. Subsequently, the most similar document is compared in detail with the target document using a fine-grained approach. As a result, only a few selected documents undergo a thorough comparison in terms of sentence and document similarity. Consequently, the proposed method incurs a low comparison cost. The performance results demonstrate that the proposed *JCF* outperforms existing mechanisms in various aspects. Different from existing studies, the proposed mechanism does not directly compare documents with each other, as it would be inefficient. Instead, a cost-effective approach was taken in this study. Future work will handle document similarity comparison in multilingual settings. We will investigate techniques to handle language-specific characteristics and develop strategies for cross-lingual similarity comparisons.

Author contributions Conceptualization was performed by CYC, SJJ;methodology by CYC, SJJ, SJW;formal analysis by CYC, SJJ;writing—original draft preparation—by CYC, SJJ;writing—review and editing—by DSR. All authors have equally contributed, and all authors have read and agreed to the published version of the manuscript.

Funding This study was not funded by any institution.

Data availability The data sets generated during the current study are not publicly available but are available from the corresponding author.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Informed consent All participating authors have been informed.

Ethical approval This study does not involve either human subjects or animals.

References

1. Kabra B, Nagar C (2023) Convolutional neural network based sentiment analysis with TF–IDF based vectorization. J Integrated Sci Technol 11(3):503–503

2. Abid MA, Mushtaq MF, Akram U, Abbasi MA, Rustam F (2023) Comparative analysis of TF–IDF and loglikelihood method for keywords extraction of twitter data. Mehran Univ Res J Eng Technol 42(1):88–94
3. Sharma A, Kumar S (2023) Ontology-based semantic retrieval of documents using Word2Vec model. Data Knowl Eng 144:1–18
4. Jaca-Madariaga M, Zarrabeitia-Bilbao E, Rio-Belver RM, Moens MF (2023) Sentiment analysis model using Word2Vec, Bi-LSTM and attention mechanism. IoT Data Sci Eng Manage 160:239–244
5. Zim SK, Ashraf F, Iqbal T, Islam MA, Polok IK, Ahmed L, Mukta MSH (2023) Exploring Word-2Vec embedding for sentiment analysis of Bangla raw and romanized text. Proc Int Conf Data Sci Appl 2:677–691
6. Aoumeur NE, Li Z, EM Alshari (2023) Improving the polarity of text through word2vec embedding for primary classical arabic sentiment analysis. Neural processing letters, pp 1–16
7. Suleiman D, Awajan A, Al-Madi N (2017) Deep learning based technique for plagiarism detection in Arabic texts. In: International Conference on New Trends in Computing Sciences (ICTCS), pp 216–222
8. Luo Q, Xu W (2014) A study on the CBOW model's overfitting and stability. Association for Computing Machinery, pp 9–12
9. Shi T, Li X, Liu Z, Wang L (2022) Research on Bi-LSTM machine reading comprehension algorithm based on attention mechanism. J Phys Conf Ser 2258:1–8
10. Jing S, Liu X, Gong X, Tang Y, Xiong G (2022) Correlation analysis and text classification of chemical accident cases based on word embedding. Process Saf Environ Prot 158:698–710
11. Styawati S, Nurkholis A, Aldino A, Samsugi S, Suryati E, Cahyono RP (2022) Sentiment analysis on online transportation reviews using Word2Vec text embedding model feature extraction and support vector machine (SVM) algorithm. International Seminar on Machine Learning, Optimization, and Data Science (ISMODE), pp 163–167
12. Rahutomo F, Kitasuka T, Aritsugi M (2012) Semantic cosine similarity. Int Stud Conf Adv Sci Technol ICAST 4(1):1
13. Xia P, Zhang L, Li F (2015) Learning similarity with cosine similarity ensemble. Inf Sci 307:39–52
14. Bohra A, Barwar N (2022) A deep learning approach for plagiarism detection system using BERT. In: Congress on Intelligent Systems, pp. 163–174
15. Xia C, He T, Li W, Qin Z, Zou Z (2019) Similarity analysis of law documents based on Word2Vec. In: International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp 354–357
16. Harris ZS (1954) Distributional structure. Word 10(2–3):146–162
17. Zhang Y, Jin R, Zhou Z-H (2010) Understanding bag-of-words model: a statistical framework. Int J Mach Learn Cybern 1(1):43–52
18. Rosu R, Stoica AS, Popescu PS, Mihăescu MC (2021) NLP based deep learning approach for plagiarism detection. In: RoCHI-International Conference on Human-Computer Interaction, Romania, pp 48–60
19. Yalcin K, Cicekli I, Ercan G (2022) An external plagiarism detection system based on part-of-speech (POS) tag N-grams and word embedding. Expert Syst Appl 197:1–16
20. Awale N, Pandey M, Dulal A, Timsina B (2020) Plagiarism detection in programming assignments using machine learning. J Artif Intell Capsul Netw 2(3):177–184
21. Ramadhanti NR, Mariyah S (2019) Document similarity detection using indonesian language Word-2Vec model. In: International Conference on Informatics and Computational Sciences (ICICoS), pp 1–6
22. Qurashi AW, Holmes V, Johnson AP (2020) Document processing: methods for semantic text similarity analysis. In: International Conference on INnovations in Intelligent SysTems and Applications (INISTA), pp 1–6