EASR

Engineering and Applied Science Research

https://www.tci-thaijo.org/index.php/easr/index

Published by the Faculty of Engineering, Khon Kaen University, Thailand

# A new weighting scheme for document ranking based on the modified word-embedding method

Mohammad Edalatfard, Morteza Mohammadi Zanjireh* and Mahdi Bahaghighat

Computer Engineering Department, Imam Khomeini International University, Qazvin, Iran

Received 6 July 2023
Revised 4 February 2024
Accepted 13 February 2024

**Abstract**

Finding documents related to a search query or similar to a specific document is among the important duties of information retrieval. The vector space model has fundamental techniques, including the bag-of-words model and the TF-IDF model. These techniques are the main strategies for determining the documents' similarities. Another method for producing a document vector is using word vectors. Thanks to recent advancements in distributed meaning, word vectors can be created with significant volumes of unlabeled textual input, primarily through artificial neural network (ANN)-based methods. A semantic space is built using this data, and word-embedding vectors represent words in this semantic space. The present study examines various approaches for transforming word-embedded vectors into document vectors and offers a new approach. Ad-hoc retrieval is one of the information retrieval tasks to employ these techniques. In this research, the metrics of mean average precision (MAP) and normalized discounted cumulative gain (NDCG) are used to assess the algorithm, followed by comparing various approaches using these two measures. The findings of this investigation demonstrate that the suggested TAW-TFIDF method outperforms alternative weighting methodologies.

**Keywords:** Ad-hoc retrieval, Information retrieval, Weighting methods, Word embedding, Word2vec

## 1. Introduction

Finding document similarities is crucial for research and applications involving text. Due to its diverse applications, extensive research has been conducted to assess the similarity between the two papers. This research involves examining various features in the text and exploring them using algorithms with diverse criteria important in machine learning applications [1-5]. One of the applications of document similarity is in information retrieval. The concept of information retrieval is broad. In [6], information retrieval is defined as an academic field: “Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).”

One of the tasks of IR is to find documents similar to a specific document or related to a given query. This task is inherently challenging as the document, and the query may use different dictionaries, or the document may include the query words without being related to the query [7]. Document similarity can be calculated using various methods. Some of these techniques rely on the vector space model, such as the bag-of-words or the TF-IDF model. In contrast, specific approaches combine word embedding with basic models. In this paper, we will discuss these model types.

The vector space model is among the most popular and widely used models for document representation [8]. It is a model that presents documents in numerical form, widely accepted and employed as a standard whenever a numeric representation of text is required [9]. In the vector space model, words in a document are weighted based on their importance. Since not all words in a document carry the same level of importance, weights are applied to determine their importance within the document [8].

Weighting models are essential in large datasets, such as the web. However, since only a tiny fraction of the results are related to the highest score, thousands of websites may be relevant to the query keywords. It is crucial to rank the results using criteria that accommodate varying degrees of relevance [10]. One method to determine document similarity is to consider the frequency of word repetitions in the document. This method applies fundamental models like bag-of-words or TF-IDF, which are extensively applied in information retrieval and language modeling problems. Both models operate by counting the occurrences of words in each document. The bag-of-words model focuses on the frequency of each word's repetition in the document, while the TF-IDF model generates the vector of each document based on TF-IDF scores [6]. However, these two weighting methods could be more suitable for illustrating the semantic distance between documents. The main drawback of both methods is that they must effectively demonstrate the semantic distance between specific words [11].

Another area for improvement with traditional information retrieval methods is their reliance on exact word matching. Methods such as lemmatization can be employed to match more words; however, issues such as synonyms remain unresolved. Additionally, polysemy is not detected based on the context words are used. The root cause of this problem is that these models need to operate considering the context in which words are situated [12]. “Embedding” refers to a numerical representation of semantic units. Embedding is undoubtedly one of the most influential research areas in natural language processing. Encoding information in representations at the lowest level of processes, which can be easily integrated with advanced machine-learning models, is vital in advancing natural language processing [13].

*Corresponding author.
Email address: Zanjireh@eng.ikiu.ac.ir
doi: 10.14456/easr.2024.25

Embedding simplifies machine learning tasks for large datasets, such as sparse vectors. An embedding method can capture some of the meanings of the inputs by placing similar inputs close to each other in the embedding space. The vector created using embedding can be reused by different models [14]. In this paper, we examine various methods of creating document vectors by combining traditional methods with word-embedding vectors. We also introduce a novel approach for converting word vectors to document vectors. We apply these methods in an information retrieval task called ad-hoc retrieval.

The remainder of this paper is organized as follows: In Section 2, we first explain word-embedding and its benefits and then discuss related work. The implemented methods for this research are discussed in Section 3. Section 4 covers an explanation of two similarity measures used with the discussed methods, along with the presentation of experimental results. Finally, the results are discussed in Section 5.

## 2. Related works

The one-hot method is the most basic way to convert words to a vector. The one-hot method can be seen as a coder that maps words into patterns with fixed lengths of zero and one. Despite the simplicity of this method, it serves as the basis for other vector space models. One problem with the one-hot method is that using coded vectors makes it impossible to determine words' similarity [13]. Another embedding method is the use of distributed representations. Word embedding is a distributed vector representation of words in which a dense vector with small dimensions relative to the size of the words represents each word. This method trains vectors to use raw textual data [15].

Semantic connections captured by embedding individual words can enhance the efficiency of various information-retrieval tasks, e.g., document ranking and query reformulation [16]. Each word is represented as a vector during word embedding, typically limited to a few hundred dimensions. In contrast to traditional methods, where for large data sets, generated vectors have millions of dimensions. The length of dimensions in these methods depends on the number of words in the dictionary [17].

In recent years, methods that create word embedding using shallow neural networks have become widely popular and found diverse applications in various tasks related to natural text processing. The key feature of these methods is the arrangement of syntactic and semantic similar words close to each other in a multidimensional space [12]. Shallow artificial neural network (ANN)-based methods for creating word vectors require only large amounts of unlabeled textual data. They use this data to construct a semantic space in which words are represented as vectors. It has been proven that the geometric properties of this space are semantically and syntactically meaningful, with semantically or syntactically similar words tending to be close to each other in semantic space. Recently, word vectors have been the subject of intense research and efficient methods for calculating them are now available [18].

In [19], an algorithm called the word2Vec algorithm, created in 2013 by Thomas Mikolov, was presented. This algorithm provides a representation of words in which vectors demonstrate the semantic proximity of words to each other. Word vectors can determine the connection between words. Analyzing similarities at the word level allows for creating a vector at the document level by aggregating these vectors [20]. The underlying idea of the word2Vec algorithm is that words used in the same context will have the same meaning [21]. Although it has been proven that modeling based on this idea is significantly effective, this method of calculating word-embedding surpasses all models with complex structures; besides, it operates much faster than previous models. Regarding the scalability of this method, it can be employed in much larger datasets, resulting in more accurate embedding. The philosophy of the word2Vec algorithm is to teach vectors using simple and fast algorithms with large datasets. Simple models trained on large volumes of data prove more capable than complex ones trained on small data volumes [18].

Extended Boolean models, e.g., TF-IDF, rely on bag-of-word representations, which are reversed by inverting the number of occurrences of the document. Still considered robust basic methods, these models attempt to address two problems in information retrieval tasks: word dependencies and the inconsistency of the words. The first case assumes that words are independent. In comparison, the second describes the problem of ignoring semantically related words, especially when exact matching fails [15]. One challenge in using word embedding to determine the semantic similarity of texts is transitioning from word-level to text-level semantics. This problem has been extensively studied over the past few years [22], and various methods have been proposed to overcome this challenge.

Doc2Vec is a proposal for embedding at the paragraph level, introduced by the research team responsible for the word2Vec algorithm. Although doc2Vec is well-known, the algorithm has yet to prove very effective, mainly because of improving its performance in comparing cosine similarity with classical methods. Instead, the embedded document has been more helpful in creating feature vectors for use in deep-learning algorithms [15, 22].

Kusner et al. [11] proposed the Word's Mover distance algorithm, a similarity measure between documents based on word embedding. This method formulates an optimization problem for the minimum cost of transferring words between two documents using the earth mover distance. The cost of transferring from one word to another is the cosine distance of their reciprocal word vectors. In contrast, most document-matching methods focus on combining word embedding and creating document vectors using them. This method provides a distance function that operates directly on two word-embedding sets [9]. In [23], a vector method of converting words to document vectors is proposed. The sum of word vectors creates the document-level vector. The authors applied their method in information retrieval and demonstrated that it outperforms the Latent Semantic Indexing method, but falls short against TF-IDF and other types. In [15], it is shown that word-embedding can be used for information retrieval using the sum of word-embedding and IDF weighting. Furthermore, the proposed method is competitive with traditional methods such as TF-IDF.

The present study aims to find a way to detect semantic similarity between the query and the document by combining fundamental methods and word embedding. Also, it tries to achieve the highest performance based on implemented criteria. To this end, we implement and study different methods of combining basic models with word-embedding, such as those presented in [15] and [23].

## 3. Methodology

This section presents our proposed method. One of the ways to obtain document-level vectors is by combining word vectors, similar to the methods presented in [15] and [23]. The first method to obtain a vector for each document is to calculate the average of the word-embedding vectors within that document. This method calculates the average of the embedding vectors of the words in the document.

Suppose a matrix of the word-embedding vectors inside the vocabulary called E, with dimensions m × h. In this matrix, the length of each embedding vector is h for each word, and the size of the dictionary is equal to m. Additionally, there is matrix B that represents word-document occurrences. The values in this matrix are either 0 or 1. If a word exists in a document, its value is 1; otherwise, it is 0. The dimensions of this matrix are n × m, where m represents the number of words in the vocabulary, and n represents the number of documents in the dataset. Eq. (1) is used to calculate the vector per document, equal to multiplying the matrix E by B. This paper refers to this method as the Average of Weightings (AW) method for short.

$$W = B \cdot E,\ W \in Rn \times h \tag{1}$$

Another method for similarity detection is using the number of word repetitions in documents as a weighting pattern for word-embedding vectors. In this method, the embedding vector of each word is multiplied by the number of repetitions of the word in the document.

Suppose a matrix of word-document occurrences called C with dimensions n × m, where m represents the number of words in the vocabulary and n represents the number of documents in the document set. Additionally, there is a matrix called E, consisting of word-embedding vectors inside words, with dimensions m× h. The value of $C_{ij}$ represents the number of occurrences of the word j in document i. The row j inside the matrix W corresponds to column j of matrix C. Eq. (2) calculates the weight matrix by multiplying two matrices, C and E.

$$W = C \cdot E,\ W \in Rn \times h \tag{2}$$

The following method involves using the IDF of each word to weigh the word-embedding vectors. In this method, the embedding vector of each word is multiplied by the IDF score of that word in the document.

Suppose a matrix C that represents the word-document occurrence matrix, with dimensions n × m. Additionally, we have a matrix called E, containing the word-embedding vectors within the vocabulary. If the word j occurs in document i, the value of $C_{ij}$ will be one; otherwise, it will be zero. We also have the matrix F, representing the weight vector of the IDFs of the words in the set of documents, with dimensions m × m. Eq. (3) is used to calculate matrix I, where each row represents the word weight of a document based on IDF.

$$I = C \cdot F,\ I \in Rn \times m \tag{3}$$

Eq. (4) is used to calculate the weight of the document matrix, which is the product of matrix I multiplied by matrix E. This method is called the Average of Weightings and IDF (AW-IDF) for short.

$$W = I \cdot E,\ W \in Rn \times h \tag{4}$$

The following method involves using the TF-IDF of each word to weight the word embedding vectors. In this method, we multiply the embedding vector of each word by the TF-IDF of the word in the document.

Suppose we have a matrix T containing the TF-IDF weight of the word-document pairs, with dimensions n×m. Here, m represents the number of words in the vocabulary, and n represents the number of documents in the document set. Moreover, we have an E matrix consisting of the word-embedding vectors within the words, with dimensions m×h. The value of $X_{ij}$ represents the weight of the word j and in document i. Eq. (5) computes the document weight matrix, which results from the element-wise multiplication of matrix T and matrix E. This method is called the Average of Weightings and TF-IDF (AW-TFIDF) for short.

$$W = T \cdot E,\ W \in Rn \times h \tag{5}$$

In addition to the mentioned methods, we introduce a new weighting method. For each document, the k initial words of the document with the highest TF-IDF weights are considered to calculate the score without calculating the weight of the remaining words. Suppose matrix F, where the words with the highest TF-IDF weights are arranged. Here, k words from the initial document with the highest TF-IDF scores are chosen for calculation, resulting in a matrix with dimensions n×k. Besides, we have a matrix of word-embedding vectors called E, with dimensions k×h. Eq. (6) is used to calculate the weight of the document matrix by the inner product of the matrices of F and E.

$$W = F \cdot E,\ W \in Rn \times h \tag{6}$$

In this method, the document vector is obtained using the TF-IDF weight method, followed by sorting the words in order by the highest TF-IDF weight and choosing the initial k words from each document. Various algorithms, such as the word2Vec, can be used to obtain the word-embedding vector. In this method, the words in the dictionary are converted into vectors. Next, the weighted average of each word embedding vector is calculated using the TF-IDF score of the word. This average creates one vector per document. For short, this method is called the Truncated Average of Weightings and TF-IDF (TAW-TFIDF).

The document weight matrix is obtained using various methods presented in Figure 1. To calculate the similarity between the document and query vectors, we can use Eq. (7) [6]. This equation represents the cosine similarity relation to obtain the similarity of the queries and documents:

$$sim(d,q) = \frac{V(d) \cdot V(q)}{\left|\left|\vec{V}(d)\right|\right| \cdot \left|\left|\vec{V}(q)\right|\right|} \tag{7}$$

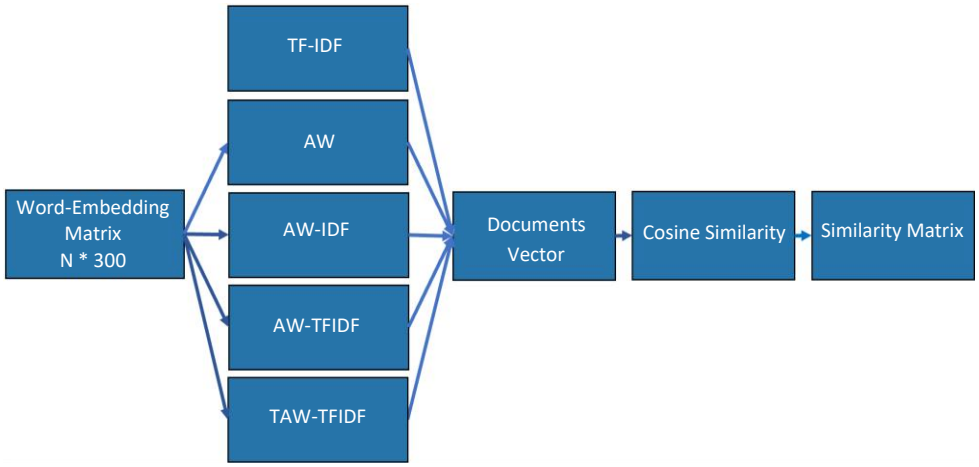


**Figure 1** Steps of creating a similarity matrix

Cosine similarity values are sorted from highest to lowest based on each query. One advantage of using word embedding in document vector creation is that, in the presence of semantic similarity between the query words and the document words, even without an exact match, the document and query vectors will still be close to each other. For instance, consider a scenario where a document contains the word "car" multiple times, and the user searches for the word "machine". If TF-IDF is employed, the document will receive a low score because "machine" is absent in the text. However, in methods based on embedding words, this document would be assigned a higher score because, in the embedding space, the representations of the words "machine" and "car" are close to each other.

## 4. Experimental results

This section describes the set of documents used in this research. Next, it explains the pre-processing steps, followed by discussing how to evaluate the proposed method. Table 1 offers the details about the hardware used for this experiment.

**Table 1** The system specifications

| **Operating System** | **Windows 10 (x64)** |
|---|---|
| CPU | Intel core i7, 8 cores |
| RAM | 16 GB |

The proposed method was evaluated using the data of the TREC 2019 dataset. This database contains three files: the first contains a set of documents, the second has a set of queries, and the third includes a set of results [24]. TREC stands for Text Retrieval Conference, dating back to 1992, providing the infrastructure for large-scale evaluation of information retrieval methods. For our evaluation, we selected a database containing 367 thousand queries and 3.2 million documents to assess document ranking. In our experiments, various systems retrieve the top-ranked documents for each query from the document set, which are then re-evaluated by human agents based on the degree of relevance [25]. Due to the dataset's large volume, we randomly selected 200 queries and 20,000 related documents for evaluating the methods outlined in the previous section. The text of the dataset documents must undergo pre-processing before being used in the algorithms. Figure 2 presents the steps of pre-processing.

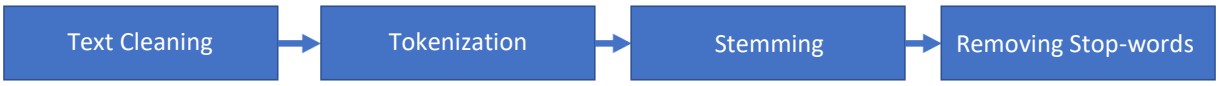


**Figure 2** Several steps of pre-processing in text mining

First, the text is cleared by tokenizing it inside each document to convert the text to a vector. Then, we stem the words and remove stop words from the tokens. Finally, extra letters are removed and all uppercase letters are converted to lowercase to clear the text.

During the tokenization process, documents are converted into separate units. The SpaCy tool is used for tokenization, stemming, and stop word removal. Numerous information retrieval studies have shown that retrieval system users focus on results with the highest scores. Therefore, data retrieval criteria focus on comparing retrieved results with ideal results. These criteria are ideally generated manually or based on user behavior feedback. They are typically calculated at the ranking position, and their averages are calculated for all queries [25].

This section describes the criteria used to evaluate the retrieval system. The standard information retrieval system evaluation involves distinguishing between related and unrelated documents. Accordingly, it is possible to discern the documents relevant to the query and the unrelated ones [8]. All criteria require a set of related (gold) documents and a list of documents the system returns using the query. Performance is evaluated using ranking criteria such as Mean Average Precision and normalized discounted cumulative gain [15]. First, precision is defined; suppose we have a model that labels one set of samples positively and another negatively. The precision measurement of the efficiency of our model is calculated using Eq. (8):

$$\text{Precision} = \frac{TruePositive}{TruePositive+FalsePositive} \quad (8)$$

Precision and recall criteria do not consider the order of the list returned by the system. However, considering the order is crucial in information retrieval; i.e., where related documents should be at the top of the list returned by the search engine. To assess the accuracy of the information retrieval model, the precision criterion is calculated at a specific rank. This precision at a particular rank is denoted as P@K, where K is the rank at which the correct criterion is calculated. Average precision is then calculated as the average of precision values across all ranks up to the specified K. The average precision is calculated for each query, and for a ranked list of documents, it is determined using Eq. (9) [25]. The average precision considers the position of the returned documents on the list when calculating the performance score for a query [12].

$$\text{Ave}P_q = \frac{\sum_{i,d\in R} Precision_{q,1} \times rel_q(d)}{\sum_{d\in D} rel_q(d)} \tag{9}$$

For a set of queries, the Mean Average Precision is calculated by calculating the Average Precision for each query. The Mean Average Precision is defined across all queries [10, 25].

Another criterion for assessing the retrieval system is the normalized discounted cumulative gain. This criterion is applicable when there is a graded relationship for all queries. The Mean Average Precision criterion evaluates documents in a Boolean way. This criterion makes no distinction between a document that is very relevant to a query and a less relevant document. Thus, it distinguishes between relevant and unrelated documents without considering the degree of relevance. In contrast, the NDCG criterion considers the score of related documents in its calculation [8]. DCG is computed using Eq. (10) [14]:

$$DCG_q \sum_{(i,d)\in R_q} \left(\frac{2^{rel_q^{(d)}}-1}{\log_2(i+1)}\right) \tag{10}$$

The ideal DCG is calculated similarly to Eq. (9), but assuming that the ideal ranking is returned as the response. Finally, the normalized discounted cumulative gain is calculated using Eq. (11) [10, 25].

$$NDCG_q = \frac{DCG_q}{IDCG_q} \tag{11}$$

In this experiment, we used the Scikit-learn library to convert the explicit texts into TF-IDF and IDF vectors and the Gensim library to create a word-embedding matrix.

Table 2 compares the results of the traditional TF-IDF weighting model with the weighting model based on TAW-TFIDF embedding for ten queries. As can be seen, the embedding-based method yields better results according to the AP and NDCG criteria.

**Table 2** Results comparing the traditional TF-IDF weighting model with the TAW-TFIDF embedding-based model for ten queries

| Query No | Taw-TFIDF | | TF-IDF | |
|---|---|---|---|---|
| | NDCG | AP | NDCG | AP |
| 480210 | 0.517 | 0.471 | 0.469 | 0.426 |
| 260536 | 0.699 | 0.739 | 0.734 | 0.714 |
| 39422 | 0.578 | 0.497 | 0.511 | 0.442 |
| 79798 | 0.54 | 0.468 | 0.493 | 0.461 |
| 92082 | 0.751 | 0.554 | 0.515 | 0.318 |
| 120204 | 0.399 | 0.341 | 0.256 | 0.256 |
| 510587 | 0.792 | 0.727 | 0.772 | 0.765 |
| 829202 | 0.427 | 0.433 | 0.368 | 0.351 |
| 7957 | 0.33 | 0.3 | 0.181 | 0.232 |
| 947184 | 0.449 | 0.431 | 0.477 | 0.415 |

Table 3 compares the results of the AW-TFIDF weighting model with the TAW-TFIDF weighting model for ten queries. As can be seen, the TAW-TFIDF method outperforms the AP and NDCG criteria.

**Table 3** Results of comparing the TAW-TFIDF weighting model with the AW-TFIDF weighting model for ten queries

| Query No | Taw-TFIDF | | AW-TFIDF | |
|---|---|---|---|---|
| | NDCG | AP | NDCG | AP |
| 480210 | 0.517 | 0.471 | 0.369 | 0.338 |
| 591004 | 0.699 | 0.739 | 0.636 | 0.594 |
| 260536 | 0.578 | 0.497 | 0.51 | 0.419 |
| 39422 | 0.54 | 0.468 | 0.202 | 0.255 |
| 400614 | 0.751 | 0.554 | 0.143 | 0.106 |
| 79798 | 0.399 | 0.341 | 0.37 | 0.356 |
| 92082 | 0.792 | 0.727 | 0.771 | 0.687 |
| 210204 | 0.427 | 0.433 | 0.389 | 0.358 |
| 133639 | 0.33 | 0.3 | 0.202 | 0.241 |
| 510587 | 0.449 | 0.431 | 0.456 | 0.351 |

Figures 3 and 4 present a graphical comparison of the results for the three weighting models TF-IDF, AW-TFIDF, and TAW-TFIDF across ten queries. Figure 3 graphically compares the values of the weighting models in Tables 2 and 3 based on AP@20 criteria and Figure 4 graphically compares the values of the weighting models in Tables 2 and 3 based on the NDCG@20 criteria.

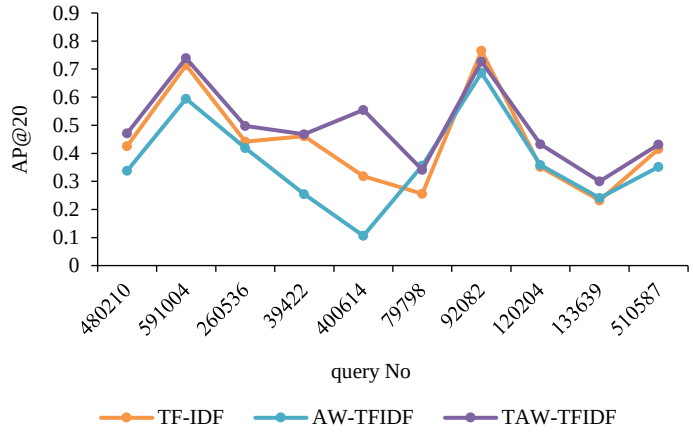


**Figure 3** Results of three weighting models: TF-IDF, AW-TFIDF, and TAW-TFIDF based on AP@20


**Figure 4** Results of three weighting models: TF-IDF, AW-TFIDF, and TAW-TFIDF based on NDCG

Tables 4, 5, and 6 present the results of comparing weighting methods for the first 30 results. The weighting methods are compared based on the criteria of MAP and NDCG for the initial ten results. As can be seen, the AW method yields the lowest scores, while the average embedding with IDF outperforms the previous model by giving weight to words with higher IDF scores. The average embedding algorithm with TF-IDF performs better than the IDF algorithm because it considers the number of word repetitions in weighting and the IDF score. Finally, the TAW-TFIDF method outperforms other weighting methods. Figures 5 and 6 present graphically the information in Tables 4-6. Figure 5 graphically compares weighting methods based on the MAP@10, MAP@20 and MAP@30. Figure 6 graphically compares weighting methods based on the NDCG@10, NDCG@20 and NDCG@30.

**Table 4** Comparing weighting methods for the first 10 results

| Method | MAP@10 | NDCG@10 |
|---|---|---|
| TF-IDF | 0.371 | 0.384 |
| AW | 0.197 | 0.179 |
| AW-IDF | 0.206 | 0.188 |
| AW-TFIDF | 0.325 | 0.324 |
| TAW-TFIDF | 0.397 | 0.397 |

**Table 5** Comparing weighting methods for the first 20 results

| Method | MAP@20 | NDCG@20 |
|---|---|---|
| TF-IDF | 0.445 | 0.479 |
| AW | 0.285 | 0.307 |
| AW-IDF | 0.297 | 0.313 |
| AW-TFIDF | 0.409 | 0.429 |
| TAW-TFIDF | 0.461 | 0.495 |

**Table 6** Comparing weighting methods for the first 30 results

| Method | MAP@30 | NDCG@30 |
|---|---|---|
| TF-IDF | 0.518 | 0.555 |
| AW | 0.361 | 0.413 |
| AW-IDF | 0.378 | 0.42 |
| AW-TFIDF | 0.493 | 0.519 |
| TAW-TFIDF | 0.541 | 0.571 |

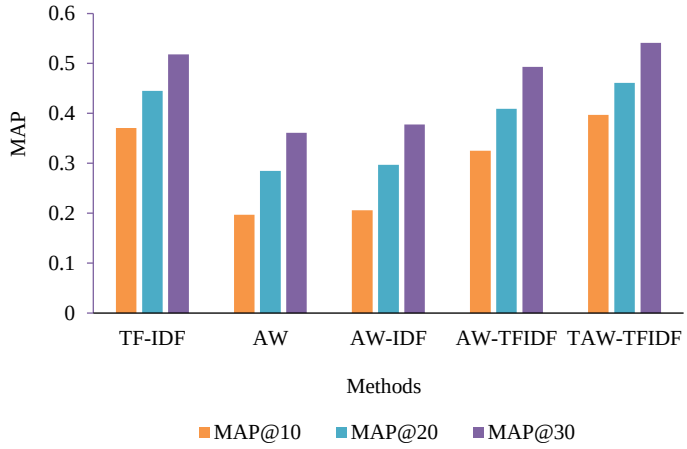


**Figure 5** A comparison of weighting methods based on the MAP

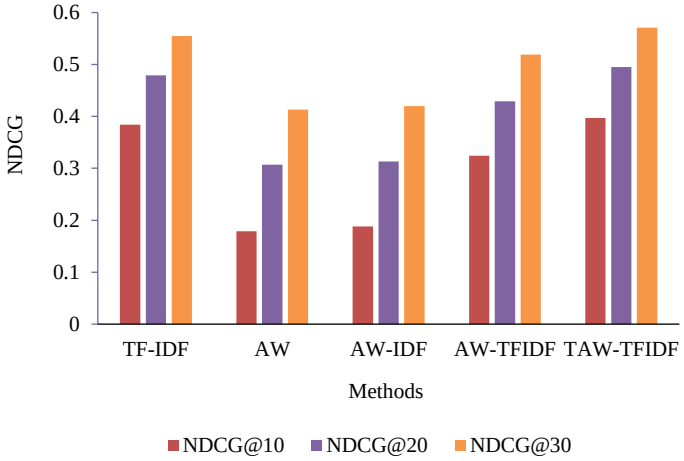


**Figure 6** A comparison of weighting methods based on the NDCG

## 5. Conclusion

This paper investigates various word-embedding vector creation techniques and applies them to the Ad-hoc information retrieval challenge. To this end, MAP and NDCG criteria were applied to compare and contrast different approaches and evaluate the algorithm.

According to the MAP@30 criteria, AW-TFIDF scored 0.493, whereas our novel TAW-TFIDF attained a value of 0.541, indicating a 10% relative improvement. The TF-IDF baseline reached a value of 0.518, reflecting a 4.5% relative improvement. Analyzing the results revealed that using an average word embedding to create the document embedding yielded the lowest performance. This achievement is attributed to the uniform weight assigned to all words, treating less important words equally with more significant ones, resulting in an inadequate document representation.

The following method, utilizing IDF as weight, demonstrated improved performance. Meanwhile, employing TF-IDF as weight yielded even better results. Lastly, weighting according to the TF-IDF method and truncating it proved to outperform the other methods. This achievement is because words with lower TF-IDF scores in the document are excluded from the scoring, enhancing the document vector's representation. The proposed method competes with the primary TF-IDF method and outperforms it. A limitation of this method is that it has been evaluated on a sample dataset. Thus, its effectiveness could be further assessed in larger datasets and compared with other methods. In this respect, investigating the impact of changing the constant 'k' for selected words on document vectors could provide valuable insights for future research.

Although the amount of calculations has decreased in the proposed method, the accuracy of finding related documents has increased, resulting in an average performance improvement of 10% compared to the AW-TFID model and 4.5% compared to the TF-IDF model.

While the present study employed various word-embedding vector construction strategies, it showed encouraging results in the ad-hoc information retrieval task. Nevertheless, there is still ample room for improvement. Many natural language processing tasks have been revolutionized by deep learning, holding the potential for a significant impact on our research. Deep learning could enhance the efficiency of ad-hoc retrieval systems. Future studies should focus on integrating our proposed TAW-TFIDF method with state-of-the-art deep learning techniques. Leveraging the power of robust deep learning models [26-30] can extend the boundaries of information retrieval, unlocking new possibilities for more accurate and efficient document retrieval applications.

## 6. References


[1] Gomaa WH, Fahmy AA. A survey of text similarity approaches. Int J Comput Appl. 2013;68(13):13-8.
[2] Sitikhu P, Pahi K, Thapa P, Shakya S. A comparison of semantic similarity methods for maximum human interpretability. 2019 Artificial intelligence for transforming business and society (AITB); 2019 Nov 5; Kathmandu, Nepal. USA: IEEE; 2019. p. 1-4.
[3] Khorasani F, Zanjireh MM, Bahaghighat M, Xin Q. A tradeoff between accuracy and speed for k-means seed determination. Comput Syst Sci Eng. 2022;40(3):1085-98.
[4] Amouee E, Zanjireh MM, Bahaghighat M, Ghorbani M. A new anomalous text detection approach using unsupervised methods. Facta Univ Electron Energ. 2020;33(4):631-53.
[5] Bozorgi M, Zanjireh MM, Bahaghighat M, Xin Q. A time-efficient and exploratory algorithm for the rectangle packing problem. Intell Autom Soft Comput. 2022;31(2):885-98.
[6] Manning CD. An introduction to information retrieval. United Kingdom: Cambridge University Press; 2009.
[7] Mitra B, Nalisnick E, Craswell N, Caruana R. A dual embedding space model for document ranking. arXiv:1602.01137. 2016:1-10.
[8] Plansangket S. New weighting schemes for document ranking and ranked query suggestion [dissertation]. England: University of Essex; 2017.
[9] Alvarez JE. A review of word embedding and document similarity algorithms applied to academic text [thesis]. Germany: University of Freiburg; 2017.
[10] Aggarwal CC. Machine learning for text: an introduction. USA: Springer International Publishing; 2018.
[11] Kusner M, Sun Y, Kolkin N, Weinberger K. From word embeddings to document distances. Proceedings of the 32nd international conference on machine learning; 2015 Jun 1; Lille, France. United States: MLR Press; 2015. p. 957-66.
[12] George B. Document reranking with deep learning in information retrieval [thesis]. Greece: Athens University of Economics; 2018.
[13] Garcia M. Embeddings in natural language processing: theory and advances in vector representations of meaning. Comput Linguist. 2021;47(3):699-701.
[14] Jayashree R, Christy A. Improving the enhanced recommended system using Bayesian approximation method and normalized discounted cumulative gain. Procedia Comput Sci. 2015;50:216-22.
[15] Galke L, Saleh A, Scherp A. Word embeddings for practical information retrieval. INFORMATIK 2017. Deep Learning in heterogenen Datenbeständen; 2017 Sep 25-29; Chemnitz, Germany. Bonn: Gesellschaft für Informatik; 2017. p. 2155-67.
[16] Roy D, Ganguly D, Bhatia S, Bedathur S, Mitra M. Using word embeddings for information retrieval: how collection and term normalization choices affect performance. Proceedings of the 27th ACM international conference on information and knowledge management; 2018 Oct 17; Torino, Italy. New York: ACM; 2018. p. 1835-8.
[17] Singh TD. Combined word and network embeddings: an analysis framework of user opinions on social media [dissertation]. USA: The University of North Carolina; 2020.
[18] Goldberg Y. Neural network methods for natural language processing. USA: Springer Nature; 2022.
[19] Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems 26 (NIPS 2013). USA: Curran Associates Inc; 2013. p. 3111-9.
[20] Wu L, Yen IEH, Xu K, Xu F, Balakrishnan A, Chen PY, et al. Word mover's embedding: from word2vec to document embedding. arXiv:1811.01713. 2018:1-15.
[21] Church KW. Word2Vec. Nat Lang Eng. 2017;23(1):155-62.
[22] Le Q, Mikolov T. Distributed representations of sentences and documents. Proceedings of the 31st International Conference on Machine Learning; 2014 Jun 18; Lille, France. United States: MLR Press; 2014. p. 1188-96.
[23] Clinchant S, Perronnin F. Textual similarity with a bag-of-embedded-words model. Proceedings of the 2013 Conference on the Theory of Information Retrieval; 2013 Sep 29 - Oct 2; Copenhagen, Denmark. New York: ACM; 2013. p. 117-20.
[24] Craswell N, Mitra B, Yilmaz E, Campos D, Voorhees EM. Overview of the TREC 2019 deep learning track. arXiv:2003.07820. 2020:1-22.
[25] Mitra B, Craswell N. An introduction to neural information retrieval. Boston: Now Publishers; 2018.
[26] Bahaghighat M, Xin Q, Motamedi SA, Zanjireh MM, Vacavant A. Estimation of wind turbine angular velocity remotely found on video mining and convolutional neural network. Appl Sci. 2020;10(10):3544.
[27] Bahaghighat M, Abedini F, Xin Q, Zanjireh MM, Mirjalili S. Using machine learning and computer vision to estimate the angular velocity of wind turbines in smart grids remotely. Energy Reports. 2021;7:8561-76.
[28] Ghorbani M, Bahaghighat M, Xin Q, Özen F. ConvLSTMConv network: a deep learning approach for sentiment analysis in cloud computing. J Cloud Comp. 2020;9:16.
[29] Abedini F, Bahaghighat M, S'hoyan M. Wind turbine tower detection using feature descriptors and deep learning. Facta Univ Electron Energ. 2020;33(1):133-53.
[30] Hajikarimi A, Bahaghighat M. Optimum outlier detection in internet of things industries using autoencoder. In: Khosravy M, Gupta N, Patel N, editors. Frontiers in Nature-Inspired Industrial Optimization. Springer Tracts in Nature-Inspired Computing. Singapore: Springer; 2022. p. 77-92.