American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-6, Issue-7, pp-251-256
www.ajer.org
Research Paper Open Access

Comparison of the Documents Based On Vector Model: A Case Study of Vietnamese Documents

*Hung Vo Trung[1], Ngoc Anh Nguyen[1], Hieu Ho Phan[1], Thi Dung Dang[2]
[1]*The University of Danang, Vietnam;*
[2]*Mien Tay Construction University*
Corresponding Author: Hung Vo Trung

Abstract: *In this paper, we present the result of the study related to the comparability of two documents. This comparison aims to determine the similarity of a text/document with other one. Our method is converting a document into vector. Each element of vector is a weight corresponding to the index term that appears in the text. The similarity comparison of the two texts are transformed into angle created by two vectors. This angle represents the similarity/difference between the two documents. We have developed a tool that compares an analysis document with two or set of documents. The results reflect exactly the similarity/difference and achievement of the objectives.*

Keywords: *Vector model, document comparison, copy detection, measurement, vectorization*

Date of Submission: 12-07-2017 Date of acceptance: 22-07-2017

I. INTRODUCTION

Along with the development of the Internet, exchanges and sharing of documents are also common. The publishing of articles, research papers, internship reports, graduation thesis, dissertations… are increasingly popular on the internet. Users can find the information they need relatively quickly and easily. However, besides the advantage of providing a rich source of reference material, plagiarism is becoming a big problem. The question proposed is how to detect the copying a text from other one, to improve the quality of research documents. Currently, the research to find duplication on the text has come up with many effective tools and can be used online such as Plagiarism Checker Software, Turnitin, etc. But these systems only serviceto English documents, there are not plagiarism checker software for Vietnamese documents. In addition, these systems don't permit users to update new data to the database. Therefore, further research is needed to find better solutions. Currently, there are many two-text matching algorithms that are widely used in various fields such as information search, intrusion detection in cybersecurity, pattern search in ADN sequences, etc. Each algorithm Matching has a different approach and each algorithm has its own advantages and limitations [1]. In this paper, we focus on improving the text comparison algorithm based on vector model. To detect whether D_1 is copied from D_2or not, the way to do this is to convert D_1 to an n-dimensional vector in which each dimension of the vector can be a word, a sentence or a paragraph in the document D_1. Similarly, convert the text of D_2 to a vector and then compare the two vectors together. This vector model is consistent with the problem of copy detection. We can evaluate the similarity of a text with many other texts available. The content of the paper is organized into 5 parts. The first part presents reasons for studying and an overview about methods and results. The second part presents some research results related including the vector model and text matching. Part 3 introduces the content of our proposed solutions related to the generalized model, the process of document vectorization and some related algorithms. Part 4 presents the results of the experiment and some comments on the results. The last part is the conclusion and development direction in the future.

II. RELATED WORKS

1. Vector model

The vector model is a popular and simple algebra model used to represent a text. A text is described by a set of keywords, also known as index terms, after removing the words (stop word). A set of index words/terms defines as a vector where each index wordis representedas a dimension in the vector. These index words are words that contain the body of the text, each of which is assigned a weight. You can use mathematical operations on the vector model to compute the similar measure between the query text and the sample text. [7]

[9] For example, the text *d* is represented in the form with an m-dimensional vector. Where m is the dimension of the text vector *d*, each dimension corresponds to a word in the set of words, w_i is the weight of the i^{th} character (with $1 \leq i \leq m$). The similarity of the two texts is often defined as the distance between the points or the angle between the vectors in vector space.

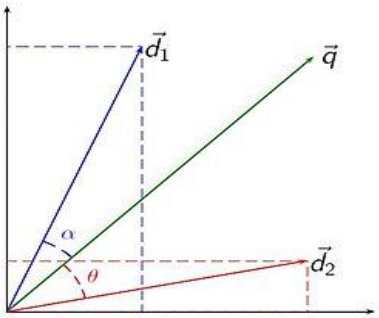

Figure 1. Example of angle created by two vectors $\vec{d_1}$, $\vec{d_2}$ with $\vec{q}$

2. String matching

The string matching problem is expressed as follows: Given a string with the length of*n* and *a*is a patternwith the length of *m*, find the occurrence of the pattern in the string. To find all the occurrences of the pattern in the string, do this by scanning through the entire string sequentially. The "match" problem is characterized as a search problem, in which the pattern is treated as a key.
Currently, there are several algorithms to solve the matching problem:

a. Brute-Force algorithm

The Brute-Force algorithm is a shallow-type algorithm. By moving the counter *j* from the left to the right of each character of the string. Then take *m* consecutive characters in *s* (starting at position *j*) forming an extra string *r*. Compare *r*with*pattern*, if the same, output the result. Repeat this process until *j*> *n-m + 1*. [4]

b. Knuth-Morris-Pratt algorithm

The Knuth-Morris-Pratt matching algorithm (or KMP algorithm)searches for the appearance of a "word" in a "text string" by continuing the search process when it does not fit, ignoring the process. Check the characters compared before. The idea, at each point in time, that the algorithm is always defined by two integer types, *n* is the length of the string *s*, and *m* is the length of the pattern p. [3]

c. Boyer-Moore algorithm

The idea of this algorithm is to assume that there are strings s and strings p, which need to find p in s; Start checking the characters of p and s from right to left and when detecting the first difference the algorithm will proceed to translate p to the right to make further comparisons. [2]

d. Rabin-Karp algorithm

The Rabin-Karp algorithm [5] uses the equivalence of two congruence numbers with a third number (for positive integers *n*, two integers *a* and *b* are called congruencies under module *n* if they have the same number. Balance when divided by *n*). We can see that each letter of the alphabet *A* is a number in the base system *d*, with $d = |A|$.

Given the given form *p [1 ... m]*, calling *p* represents its corresponding number. Similarly, for the text *T[1 ... n]*, we denote *t*, representing the number of substrings *T[s + 1 ... s + m]* of length *m*, where *s = 0,1 ...*, *m*n*. Obviously, $t_s = p$ if and only if *T[s + 1 ... s + m] = p [1 ... m]*.

e. Comment

We realize that searching with Brute-Force may be very slow for some templates, for example if the string to look at is a binary string. In the worst case, when all the samples are zero and end up with a number of *1*. For each position *n-m + 1* the positions can be matched, is compared with each character of the text, so it is necessary to make *n-m + 1* comparisons. On the other hand, m is very small compared to *n*, so the number of character comparisons is approximately *m * n*.

The Knuth-Morris-Pratt algorithm uses less mathematical comparisons than Brute-Force. However, in practical application, the Knuth-Morris-Pratt algorithm is not significantly faster than the Brute-Force algorithm.

The Knuth-Morris-Pratt algorithm performs a sequential search within the text and does not require backing up that text. This makes sense when applied on a large file, which consumes less caching.

Boyer-Moore algorithm does not use more than $m + n$ character comparison. In fact, when text characters do not appear in the form or except for a few being present in the template, each comparison leads to a template that will translate to the right m of the character, so for text Large and not long sample, the algorithm must use n/m steps. The Rabin-Krap algorithm is almost linear. The number of comparisons in this algorithm is $m + n$; the algorithm only looks for a location in the text that has the same hash value with the pattern.

III. SOLUTIONS PROPOSED

1. General model

The process of comparing a query text to a set of sample text is done in the following pattern:

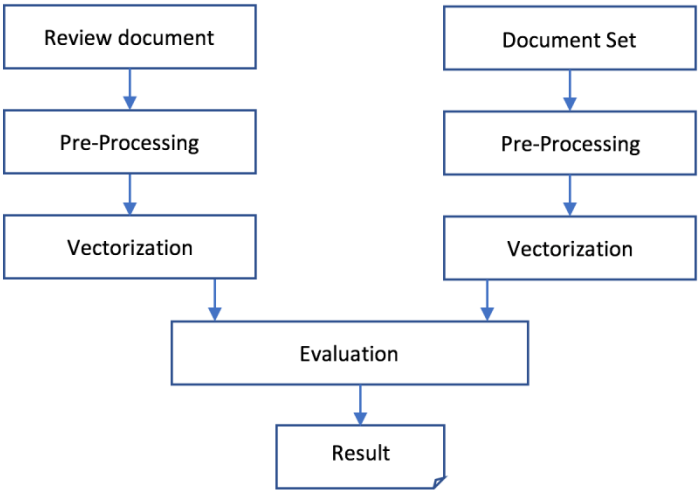

Figure 2. Comparison of two documents

According to this model, the set of sample texts must be processed [8] and vectored for storage. Then, each text that needs to be compared to the sample text will also be processed, vectored, and compared to the archived data to detect the same (copy) level from the query text to the file.

2. Vectorization model

In the course of comparison, the vectorization step is for the purpose of rendering the text in vector form for future comparisons. Vectorization can be done based on the word processor (each vector element is word) or sentence unit (each vector element is a sentence).

The process of vectorization in units of words is as follows:

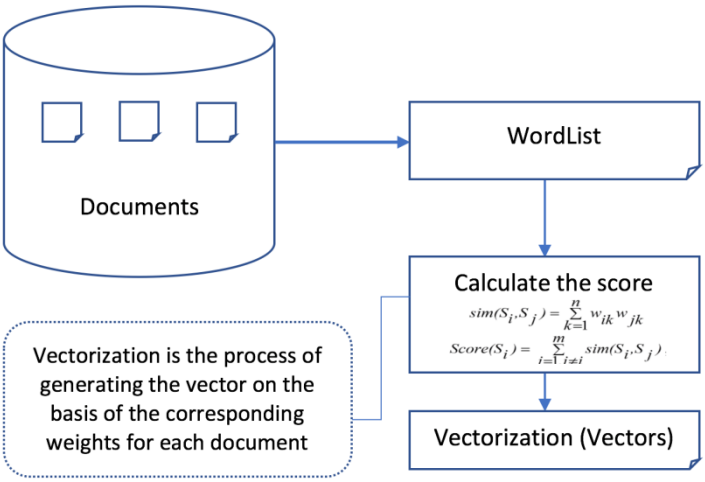

Figure 3. The process of vectorization using wordunit

3. Matching algorithms

a. Match on the unit vector model is from

The purpose of the algorithm is to calculate the similarity of the evaluation text with the given set of text samples based on the unit of words.

Input: A set of texts given and the text/document to be evaluated (already preprocessed).

Output: The same ratio between the text of the review and the given text.

Algorithm:

Step 1: Preprocessing

- Text formatting in plain text format (.txt).
- Separation of words.
- Create a Wordlist vocabulary list.
- Remove StopWord.

Step 2:

- Calculating the weight of index words W=TF*IDF*N with N = 1.
- Calculation of local weights L_{ij}:

$$TF = \begin{cases} 1+\log f_{ij} & if \quad f_{ij} > 0 \\ 0 & if \quad f_{ij} = 0 \end{cases}$$

- Calculatef_{ij}: Number of occurrences of i in document j.

TF(*i, j*) // *i: Inđex termi; j: document j*

if f_{ij}= 0 then return(0)

else return(1 + log(f_{ij}))

- Calculate the global weighting G_i:

$$IDF = \log\left(\frac{N}{n_i}\right)$$

IDF(i, docs) // docs are set of.

CalcN: number of ducuments in the set.

Calcn$_i$: number of documents which is included word.

if (n_i>0) thenreturn log(N/n_i)

else return 0

Step 3: Build a weight matrix to calculate the semantic similarity between the two texts.

$$\cos\theta = \frac{d_j{}^T q}{\|d_j\|_2 \|q\|_2} = \frac{\sum_{i=1}^{m} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{m} w_{ij}{}^2}\sqrt{\sum_{i=1}^{m} w_{iq}{}^2}}$$

Step 4: Calculate the semantic similarity between the two texts.

Step 5: Build a weighting matrix that calculates the order of similarity between two documents.

$$S_{r_j} = 1 - \frac{\|r_j - r_q\|}{\|r_j + r_q\|} = \frac{\sqrt{\sum_{i=0}^{m-1}\left(r_{ij} - r_{iq}\right)^2}}{\sqrt{\sum_{i=0}^{m-1}\left(r_{ij} + r_{iq}\right)^2}}$$

Step 6: Calculate the order of similarities between two documents.

Step 7: Measure the same measurement completely between two documents.

$$S(d_j, q) = \delta S_{\cos\theta_j} + (1-\delta) S_r = \delta\left(\frac{d_j{}^T q}{\|d_j\|_2 \|q\|_2}\right) + (1-\delta)\left(\frac{\|r_j - r_q\|}{\|r_j + r_q\|}\right)$$

b. Match on the unit vector model is the sentence

The purpose of the algorithm is to calculate the similarity of the evaluation text with a given set of sample text based on unit of sentence.

Input: A set of texts given and the text / text to be evaluated (already preprocessed).

Output: The same ratio between the text of the review and the given text.

Step 1: Preprocessing

Step 2: Calculate the weight of the words in the text of the query text:

$$w_{qk} = TF \times IDF = TF \times \log\frac{N}{n_k}$$

$$w_{jk} = TF \times IDF = TF \times \log\frac{N}{n_k}$$

Calculate the weight of the sentence in the query text:

$$sim(S_q, S_j) = \sum_{k=0}^{n-1} w_{qk} w_{jk}$$

$$Score(S_q) = asim(S_q) = \sum_{j=0, j \neq q}^{m-1} sim(S_q, S_j)$$

Step 3: Calculate the weight of the words in the sample text:

$$w_{ik} = TF \times IDF = TF \times \log \frac{N}{n_k}$$

$$w_{jk} = TF \times IDF = TF \times \log \frac{N}{n_k}$$

Step 4: Calculate the weight of the sentence in the sample text:

$$sim(S_i, S_j) = \sum_{k=0}^{n-1} w_{ik} w_{jk}$$

$$Score(S_i) = asim(S_i) = \sum_{j=0, j \neq i}^{m-1} sim(S_i, S_j)$$

Step 5: Build up the weight matrix to calculate the semantic similarity between the two texts.

Step 6: Measure semantic similarity between two texts:

$$\cos\theta = \frac{d_j^T q}{\|d_j\|_2 \|q\|_2} = \frac{\sum_{i=0}^{m-1} w_{ij} w_{iq}}{\sqrt{\sum_{i=0}^{m-1} w_{ij}^2} \sqrt{\sum_{i=0}^{m-1} w_{iq}^2}}$$

IV. EXPERIMENT AND EVALUATE

For testing, we have built a software on C # with basic functions like word processing, text conversion and matching. Data for the test is more than 100 graduated dissertations of students in the Faculty of Information Technology, University of Technology, Danang University. These essays will be processed to retain the text, other content will be discarded (images, tables, ...).

Data after converting to text:

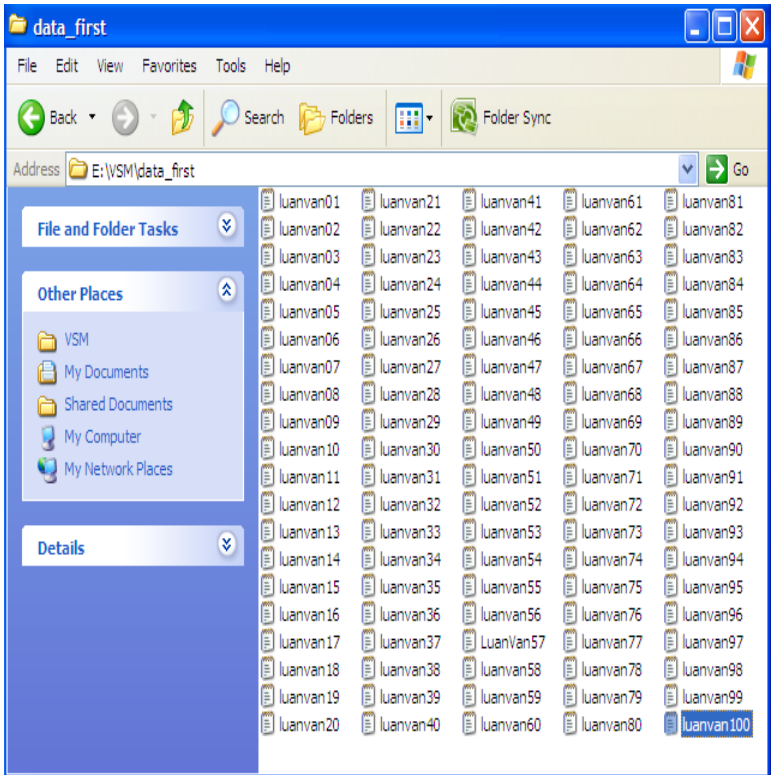

Figure 4. Data is a set of sample texts

The same ratio of documents is as follows:

Ký hiệu văn bản	Tỉ lệ giống nhau so theo từ	Tỉ lệ giống nhau so theo câu	Ký hiệu văn bản	Tỉ lệ giống nhau so theo từ	Tỉ lệ giống nhau so theo câu	Ký hiệu văn bản	Tỉ lệ giống nhau so theo từ	Tỉ lệ giống nhau so theo câu
vb01	100%	100%	vb35	45%	53%	vb69	38%	48%
vb02	51%	66%	vb36	44%	53%	vb70	39%	47%
vb03	33%	33%	vb37	50%	59%	vb71	41%	50%
vb04	41%	51%	vb38	48%	55%	vb72	42%	52%
vb05	42%	52%	vb39	45%	54%	vb73	34%	41%
vb06	43%	51%	vb40	49%	58%	vb74	39%	51%
vb07	44%	55%	vb41	46%	55%	vb75	38%	51%
vb08	42%	50%	vb42	34%	52%	vb76	44%	53%
vb09	45%	53%	vb43	40%	51%	vb77	39%	54%
vb10	43%	52%	vb44	28%	31%	vb78	46%	59%
vb11	45%	53%	vb45	39%	53%	vb79	42%	52%
vb12	44%	56%	vb46	43%	50%	vb80	43%	55%
vb13	44%	53%	vb47	42%	50%	vb81	35%	48%
vb14	46%	57%	vb48	45%	55%	vb82	37%	45%
vb15	47%	55%	vb49	47%	57%	vb83	40%	52%
vb16	48%	55%	vb50	47%	58%	vb84	33%	44%
vb17	44%	52%	vb51	46%	58%	vb85	24%	33%
vb18	47%	54%	vb52	46%	56%	vb86	35%	46%
vb19	43%	53%	vb53	48%	56%	vb87	29%	24%
vb20	47%	57%	vb54	46%	57%	vb88	26%	34%
vb21	44%	54%	vb55	46%	54%	vb89	13%	20%
vb22	44%	53%	vb56	38%	35%	vb90	29%	46%
vb23	46%	54%	vb57	46%	52%	vb91	26%	36%
vb24	44%	52%	vb58	42%	54%	vb92	37%	46%
vb25	44%	56%	vb59	34%	49%	vb93	32%	45%
vb26	46%	57%	vb60	41%	50%	vb94	33%	49%
vb27	47%	54%	vb61	47%	60%	vb95	36%	52%
vb28	48%	55%	vb62	44%	56%	vb96	39%	50%
vb29	46%	55%	vb63	44%	57%	vb97	28%	46%
vb30	45%	55%	vb64	48%	60%	vb98	40%	52%
vb31	42%	52%	vb65	52%	61%	vb99	44%	57%
vb32	44%	54%	vb66	39%	48%	vb100	47%	56%
vb33	45%	55%	vb67	40%	50%			
vb34	41%	49%	vb68	46%	53%			

Figure5. Statistics the same ratio of text 1 to other documents in the data warehouse

By experiment, we find that the match rate result has the difference between word-based vectorization and sentence-based vectorization. This difference is due to the dependence of weighting methods and functions. Comparable results are 100% when two documents are completely identical and the result is 0% when two documents do not have the same vocabulary (completely different). To get the best result when the text has a difference in length is not too large. For example, when matching a query text to a sample text, if the sample text is large and the text in the sample text is the same as the query text, it only accounts for about 16% of the sample text. Matched results run from 14% to 20%. The time and space consumed by the match process depend on the length of the match text (the number of words in the text).

V. CONCLUSION

In this paper, we have used some techniques of natural language processing, vector expressions for text representation, pattern matching algorithms, C # and semi-structured databases. The content of text/document is structuredto XML format to perform research and testing on similarity assessment between texts. We have developed tools and tested to detect duplication on text through the use of vector models. The tool allows you to check any two texts, two paragraphs of text, text with text, a text with multiple text copied together or not. The application is tested on a set of over 1000 graduated theses in the field of information technology. In the coming time, we will continue the related research: improving the vector model to limit the number of dimensions for text when vectored; Integrates preprocessing tools into the application; Study new solutions, especially the results of research in the field of biology, into the problem of copy detection.

REFERENCES

[1] J.-I. Aoe, Computer algorithms: string pattern matching strategies, *IEEE Computer Society Press*, 1994, pp 97-107.
[2] A. Postolico, R. Giancarlo, The Boyer-Moore-Galil string searching strategies revisited, *SIAM Journal on Computing*, 1986, p.p 98-105.
[3] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, *Cambridge University Press*, 1997, pp 1-58.
[4] M. Crochemore, C. Hancart, Pattern Matching in Strings, *Algorithms and Theory of Computation Handbook*, 1999, pp 11-28.
[5] D. Knuth, J.H. Morris, V. Pratt, Fast pattern matching in strings, *SIAM Journal on Computing*, 1977, pp 323–350.
[6] E. Chisholm and T.G. Kolda, New Term Weighting Formulas For The Vector Space Method In Information Retrieval, *Oak Ridge*, 1999, pp 31-63.
[7] G. Salton, A. Wong, C. S. Yang, A vector space model for automatic indexing, Communication ACM, 18, 1975, pp 613-620.
[8] L. H. Phuong and H. T. Vinh, A Maximum Entropy Approach to Sentence Boundary Detection of Vietnamese Texts, IEEE International Conference on Research, Innovation and Vision for the Future RIVF 2008, 2008, pp 102-122.
[9] N. Polettini, The Vector Space Model in Information Retrieval- Term Weighting Problem, Sommarive 14, 2004, pp 69-91.

Hung Vo Trung. "Comparison of the Documents Based On Vector Model: A Case Study of Vietnamese Documents." American Journal of Engineering Research (AJER) 6.7 (2017): 251-56.