PT. Lentera Ilmu Publisher

Online version at https://journal.lenterailmu.com/index.php/josapen

JOSAPEN

E-ISSN: 3031-2272 (Online)

JOURNAL OF COMPUTER SCIENCE APPLICATION AND ENGINEERING

Cosine Similarity-based Plagiarism Detection on Electronic Documents

Lidia Permata Sari

Department of Information System, Palembang, Indonesia

ARTICLE INFO

Article history:
Received 10 January 2023
Revised 15 March 2023
Accepted 24 May 2023

Keywords:
Plagiarism Detection
Cosine Similarity
Electronic Documents
Academic theses
Similarity Thresholds

ABSTRACT

This study addresses the prevalent issue of plagiarism in academic theses documents, recognizing the potential for undetected similarities within various sections of documents, escaping supervisor oversight. Proposing a solution utilizing the cosine similarity method—a robust technique in natural language processing and document analysis—this research aims to mitigate plagiarism occurrences. The method's benefits, such as independence from document length and high accuracy, advocate for its adoption in plagiarism detection. The study delineates the Waterfall model employed for systematic development, showcasing its structured but inflexible nature in accommodating evolving software requirements. Additionally, the elucidation of cosine similarity mechanics elucidates its pivotal role in quantifying textual resemblance between documents. Practical demonstrations using TF-IDF vectorization and cosine similarity computation offer a step-by-step understanding of the method's implementation. System design, illustrated through UML diagrams and system interface depictions, underscores the comprehensive approach taken in creating a plagiarism detection application. Lastly, successful Black Box testing confirms the application's adherence to functional criteria, validating its efficiency in identifying potential instances of plagiarism. This study contributes significantly to addressing plagiarism concerns through a robust detection mechanism.

1. Introduction

The thesis is the final assignment that a student must complete in their lecture activities and is mandatory for every student. A student's thesis is usually the result of research that has been carried out for approximately one semester.

The theses that students work on usually still contain plagiarism and may escape the supervision of the supervisor. In a thesis, there may be similarities in the title, abstract, problems, written content, methods used, discussion, research objects, and results. Through this study, the author proposes a mechanism for detecting the similarity of several documents by comparing the contents of the documents so that it will produce a value or weight of the similarity of each thesis that has been compared. One of the uses of comparing the contents of this document is to help users group theses and enable users to find out whether the contents of one thesis are similar to other theses [1], [2].

* *Corresponding author:* Lidia Permata Sari
Email: mobil1uigm@gmail.com

Until the time this study was carried out, a student could still easily and freely copy-paste a proposal or thesis report from start to finish without the supervisor knowing. Based on this, it can be concluded that plagiarism is an action or shortcut for stealing ideas, taking work results, and claiming other people's work as one's own, without including references from the source. This act of plagiarism often occurs in the world of education, such as at the university level when writing a thesis report.

Therefore, based on the background above, through this study, the author intends to build a plagiarism detection application using the cosine similarity method. The cosine similarity method is used to test document similarity. Cosine similarity is used to calculate similarity values by equating words for words and is one of the techniques for measuring text similarity that is widely used. The advantage of cosine similarity is that it is not affected by the length and shortness of a document and has a high level of accuracy. Cosine similarity can help efforts to reduce the occurrence of plagiarism.

2. Method

Figure 1 illustrates the research methodology employed by the author to fulfill the initial objectives. The system development phase utilized in this study is the waterfall model, known for its characteristic requiring completion of each phase before advancing to the subsequent one. This approach follows a structured and linear flow within software development [3], [4], [5]. The process comprises a sequence of phases that must be finished in order, where the conclusion of each phase is reliant on the prior one. Below are several primary stages within the Waterfall model:

1. Analysis: The stage where system requirements are gathered and thoroughly understood. It involves interaction with users and stakeholders to define functional and non-functional requirements.
2. Design: After the requirements are collected, the next step is to design the system architecture. This includes designing the system structure, identifying algorithms, and preparing the necessary technical specifications.
3. Coding: This stage involves coding the software according to the specifications created at the design stage. The development team creates code based on the approved design.
4. Testing: After implementation, the system is tested to ensure that all requirements have been met and that there are no significant bugs or errors. These tests include functional, performance, and security tests.
5. Delivery/Implementation: Once the system passes all the tests, it is ready to be implemented and released into a production environment or used by end users.

A significant drawback of the Waterfall model is its challenge in accommodating frequent alterations in software development requirements. Its linear structure makes it hard to revisit a prior phase once it's completed, posing difficulties in adapting to evolving needs during the cycle.

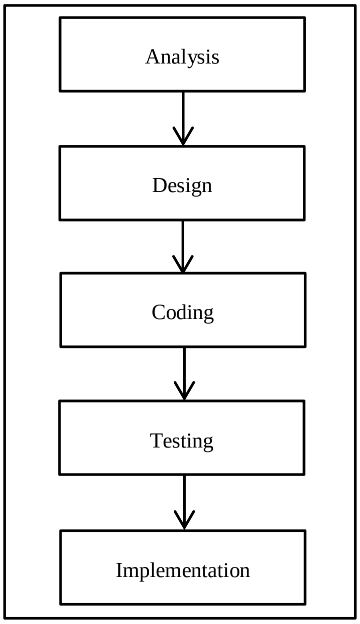


Figure 1 – Simple system development model

Cosine similarity is a metric used to measure the similarity between two non-zero vectors. It's widely applied in various fields such as natural language processing, information retrieval, and machine learning.

Here's how it works:

1. Vector Representation: When considering documents, for instance, each document is represented as a vector where each dimension corresponds to a term or a feature, and the value represents the frequency of that term in the document (in bag-of-words models) or other numerical representations (like TF-IDF - Term Frequency-Inverse Document Frequency).
2. Calculating Cosine Similarity: Cosine similarity measures the cosine of the angle between these vectors in a multidimensional space. The formula for cosine similarity between two vectors A and B is show in Equation (1):

$$Cosine\ Similarity = \frac{A.B}{\|A\|\|B\|} \quad (1)$$

Where:

- $A{\cdot}B$ represents the dot product of vectors A and B.
- $\|A\|$ and $\|B\|$ denote the Euclidean norms of vectors A and B respectively.

3. Interpretation: The resulting value lies between -1 and 1. A value closer to 1 signifies higher similarity, indicating that the vectors are more aligned or similar in the vector space. A value closer to -1 implies dissimilarity, and a value around 0 means the vectors are orthogonal or dissimilar.
4. Applications: In natural language processing, cosine similarity is often used in document similarity analysis, information retrieval (such as search engines), clustering algorithms, and recommendation systems to find similarities between documents, queries, or items based on their vectorized representations.

Cosine similarity is favored for its simplicity, efficiency in high-dimensional spaces, and effectiveness in measuring similarity between documents or items irrespective of their magnitude, focusing on the direction of the vectors.

3. Result and Discussion

The author would like to explain how to use cosine similarity to find text similarity between two documents with simple steps. Suppose we have two text documents:

Document 1: "The cat sat on the mat."
Document 2: "The dog played outside."

Step 1: Preprocess the Texts

First, preprocess the texts to make them suitable for analysis. Common preprocessing steps include lowercasing the text, removing punctuation, and tokenizing the words.

```
doc1 = "The cat sat on the mat."
doc2 = "The dog played outside."

# Preprocess the texts
doc1 = lowercase(doc1)
doc2 = lowercase(doc2)

doc1_tokens = tokenize(doc1)
doc2_tokens = tokenize(doc2)
```

Step 2: Vectorization using TF-IDF

Next, represent the text data numerically using TF-IDF vectorization. This step converts the text into numerical vectors while considering the importance of words in each document relative to their frequency across all documents.

```
# Create a vocabulary and calculate TF-IDF
all_docs = [doc1, doc2]

vectorizer = TF-IDFVectorizer()
tfidf_matrix = vectorizer.fit_transform(all_docs)

# Convert each document into TF-IDF vector representation
doc1_vector = tfidf_matrix[0]
doc2_vector = tfidf_matrix[1]
```

Step 3: Calculate Cosine Similarity

Calculate the cosine similarity between the vectors representing the documents using Equation (1). Cosine similarity measures the cosine of the angle between the vectors and ranges from -1 (completely dissimilar) to 1 (completely similar).

```
# Calculate cosine similarity between doc1 and doc2 vectors
similarity_score = cosine_similarity(doc1_vector,
        doc2_vector)

print("Cosine Similarity between the two documents:",
        similarity_score)
```

Print the cosine similarity score between the two documents, indicating their textual similarity. This pseudocode outlines the process of using cosine similarity to measure text similarity between two documents. In practice, you'd implement these steps in a programming language like Python using libraries such as scikit-learn to perform TF-IDF vectorization and cosine similarity calculations.

After understanding how the Cosine Similarity method finds text similarity between two documents, the author started designing the proposed app by creating a UML diagram [6], [7]. The proposed system design includes Use case diagrams, Activity diagrams, Sequence diagrams, and Class diagrams.

3.1. Use diagram

Several things need to be described, namely actors and use cases. Actors are users who are connected to the system and can be people (indicated by their role and not their name/personnel). The actor is symbolized by the figure of a stick man with a noun at the bottom that states the role/system. Use cases are depicted with an ellipse symbol with the name of the active verb inside which states the activity from the actor's perspective [8], [9].

3.2. Activity diagram

An activity diagram is a description of function paths in an information system [10]. In full, the activity diagram defines where the system process starts, where it stops, what activities occur during the system process, and what sequence these activities occur in.

3.3. Sequence diagram

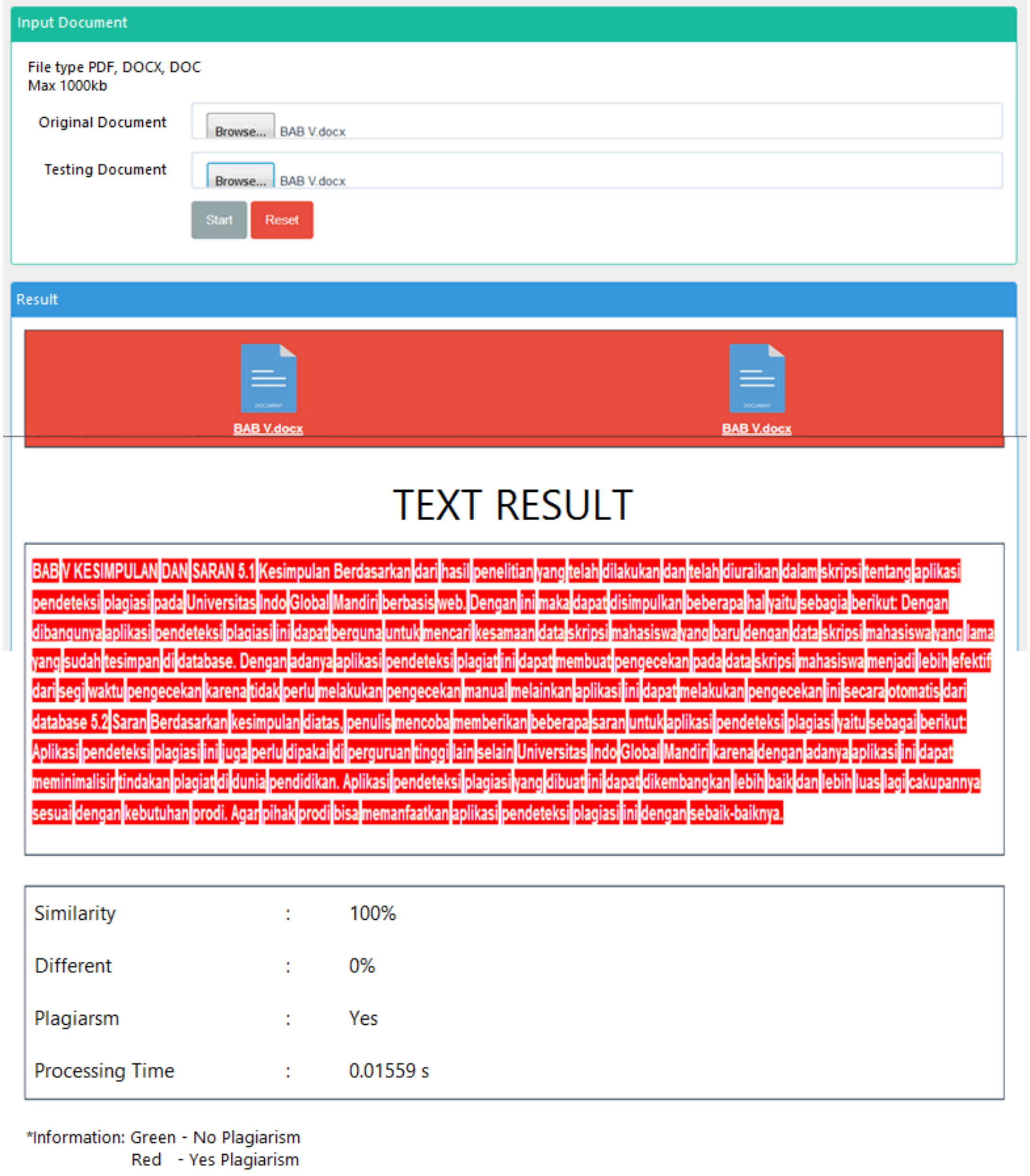
Based on the use case that has been created, a sequence diagram is obtained which describes the behavior of objects in the use case by describing the lifetime of the object and the messages sent and received between objects.

3.4. Class diagram

Class diagrams describe the types of objects in the system and the various static relationships that exist between them [11]. Class diagrams show the properties and operations of a class and the boundaries contained in the object relationships.

3.5. System Interface

A system interface refers to the point of interaction or communication between different systems, components, or software modules within a larger system or between separate systems. It defines how different parts of a system communicate, exchange data, or interact with each other. The two-document detection display is the interface that the admin uses to input two documents that will be checked for their level of similarity. These two documents consist of the original document and the test document. If the document has been entered or uploaded, the app starts processing the check and the results of similarity score will appear (Figure 2).



Input Document

File type PDF, DOCX, DOC
Max 1000kb

Original Document: Browse... BAB V.docx

Testing Document: Browse... BAB V.docx

Result

TEXT RESULT

BAB V KESIMPULAN DAN SARAN 5.1 Kesimpulan Berdasarkan dari hasil penelitian yang telah dilakukan dan telah diuraikan dalam skripsi tentang aplikasi pendeteksi plagiasi pada Universitas Indo Global Mandiri berbasis web. Dengan ini maka dapat disimpulkan beberapa hal yaitu sebagia berikut: Dengan dibangunya aplikasi pendeteksi plagiasi ini dapat berguna untuk mencari kesamaan data skripsi mahasiswa yang baru dengan data skripsi mahasiswa yang lama yang sudah tesimpan di database. Dengan adanya aplikasi pendeteksi plagiat ini dapat membuat pengecekan pada data skripsi mahasiswa menjadi lebih efektif dari segi waktu pengecekan karena tidak perlu melakukan pengecekan manual melainkan aplikasi ini dapat melakukan pengecekan ini secara otomatis dari database 5.2 Saran Berdasarkan kesimpulan diatas, penulis mencoba memberikan beberapa saran untuk aplikasi pendeteksi plagiasi yaitu sebagai berikut: Aplikasi pendeteksi plagiasi ini juga perlu dipakai di perguruan tinggi lain selain Universitas Indo Global Mandiri karena dengan adanya aplikasi ini dapat meminimalisir tindakan plagiat di dunia pendidikan. Aplikasi pendeteksi plagiasi yang dibuat ini dapat dikembangkan lebih baik dan lebih luas lagi cakupannya sesuai dengan kebutuhan prodi. Agar pihak prodi bisa memanfaatkan aplikasi pendeteksi plagiasi ini dengan sebaik-baiknya.

Similarity	:	100%
Different	:	0%
Plagiarsm	:	Yes
Processing Time	:	0.01559 s

*Information: Green - No Plagiarism
Red - Yes Plagiarism

Figure 2 – Similarity check application interface

In the end, the author carries out Black Box testing of the app that has been built. Black Box testing focuses on the functional requirements of the software [12]-[15]. Thus, black box testing allows software engineers to obtain a set of input conditions that fully utilize all functional requirements for an app. Black box testing seeks to find errors in the following criteria: Incorrect or missing functions, Interface errors, Errors in data structure or database access, and Performance errors. Based on the test results, overall the app built meets all testing criteria, in line with expectations at the start of the study.

4. Conclusion

In summary, this study delves into the pervasive issue of plagiarism within academic theses documents, acknowledging the potential for undetected similarities across various sections of documents, which could evade supervisor scrutiny. The author proposes a solution employing the cosine similarity method, a robust technique widely utilized in natural language processing and document analysis. The method's advantages, including its independence from document length and its accuracy, present a compelling case for its application in plagiarism detection. The study progresses to outline the Waterfall model adopted for system development, illustrating its structured yet inflexible nature when encountering evolving software requirements. Moreover, the explanation of cosine similarity elucidates its fundamental mechanics, showcasing its role in quantifying textual similarity between documents. The practical demonstration using TF-IDF vectorization and cosine similarity calculation provides a step-by-step insight into the methodology's application. The subsequent focus on system design, encapsulated in UML diagrams and system interface depiction, portrays the comprehensive approach undertaken in developing a plagiarism detection application. Finally, the successful Black Box testing affirms the app's adherence to functional requirements, validating its efficacy in detecting potential plagiarism instances. This holistic exploration highlights the study's contributions to tackling plagiarism concerns through a robust detection mechanism.

Acknowledgements

We would like to acknowledge Department of Information System UIGM for supporting this work.

REFERENCES

[1] S. Zouaoui and K. Rezeg, "Multi-Agents Indexing System (MAIS) for Plagiarism Detection," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 5, pp. 2131–2140, 2022, doi: 10.1016/j.jksuci.2020.06.009.

[2] Z. Liu, J. Zhu, X. Cheng, and Q. Lu, "ScienceDirect Available ScienceDirect ScienceDirect Optimized Algorithm Design for Text similarity Detection Optimized Design for Text similarity Detection Based on Algorithm Artificial Intelligence and Natural Language Based on Artificial Intelligence and Natural Language Processing Processing," *Procedia Comput. Sci.*, vol. 228, pp. 195–202, 2023, doi: 10.1016/j.procs.2023.11.023.

[3] K. D. Prasetya, Suharjito, and D. Pratama, "Effectiveness Analysis of Distributed Scrum Model Compared to Waterfall approach in Third-Party Application Development," *Procedia Comput. Sci.*, vol. 179, no. 2019, pp. 103–111, 2021, doi: 10.1016/j.procs.2020.12.014.

[4] T. Thesing, C. Feldmann, and M. Burchardt, "Agile versus Waterfall Project Management: Decision model for selecting the appropriate approach to a project," *Procedia Comput. Sci.*, vol. 181, pp. 746–756, 2021, doi: 10.1016/j.procs.2021.01.227.

[5] A. A. S. Gunawan, B. Clemons, I. F. Halim, K. Anderson, and M. P. Adianti, "Development of e-butler: Introduction of robot system in hospitality with mobile application," *Procedia Comput. Sci.*, vol. 216, no. 2019, pp. 67–76, 2022, doi: 10.1016/j.procs.2022.12.112.

[6] G. Bergström *et al.*, "Evaluating the layout quality of UML class diagrams using machine learning," *J. Syst. Softw.*, vol. 192, p. 111413, 2022, doi: 10.1016/j.jss.2022.111413.

[7] H. Wu, "QMaxUSE: A new tool for verifying UML class diagrams and OCL invariants," *Sci. Comput. Program.*, vol. 228, p. 102955, 2023, doi: 10.1016/j.scico.2023.102955.

[8] P. Danenas, T. Skersys, and R. Butleris, "Natural language processing-enhanced extraction of SBVR business vocabularies and business rules from UML use case diagrams," *Data Knowl. Eng.*, vol. 128, no. February, p. 101822, 2020, doi: 10.1016/j.datak.2020.101822.

[9] Meiliana, I. Septian, R. S. Alianto, Daniel, and F. L. Gaol, "Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm," *Procedia Comput. Sci.*, vol. 116, pp. 629–637, 2017, doi: 10.1016/j.procs.2017.10.029.

[10] Z. Daw and R. Cleaveland, "Comparing model checkers for timed UML activity diagrams," *Sci. Comput. Program.*, vol. 111, no. P2, pp. 277–299, 2015, doi: 10.1016/j.scico.2015.05.008.

[11] F. Chen, L. Zhang, X. Lian, and N. Niu, "Automatically recognizing the semantic elements from UML class diagram images," *J. Syst. Softw.*, vol. 193, p. 111431, 2022, doi: 10.1016/j.jss.2022.111431.

[12] D. Felicio, J. Simao, and N. Datia, "Rapitest: Continuous black-box testing of restful web apis," *Procedia Comput. Sci.*, vol. 219, no. 2022, pp. 537–545, 2023, doi: 10.1016/j.procs.2023.01.322.

[13] H. Bostani and V. Moonsamy, "EvadeDroid: A Practical Evasion Attack on Machine Learning for Black-box Android Malware Detection," *Comput. Secur.*, p. 103676, 2021, doi: 10.1016/j.cose.2023.103676.

[14] F. Pagano, A. Romdhana, D. Caputo, L. Verderame, and A. Merlo, "SEBASTiAn: A static and extensible black-box application security testing tool for iOS and Android applications," *SoftwareX*, vol. 23, p. 101448, 2023, doi: 10.1016/j.softx.2023.101448.

[15] C. Cronley *et al.*, "Designing and evaluating a smartphone app to increase underserved communities' data representation in transportation policy and planning," *Transp. Res. Interdiscip. Perspect.*, vol. 18, no. January, p. 100763, 2023, doi: 10.1016/j.trip.2023.100763.