DOCUMENT CLUSTERING ALGORITHM USING TF-IDF IMPLEMENTED BY HEAP

Junho Eum, Jean Paul Azzopardi

Boston University

**Table of Contents**

[***Abstract: 2***](#_heading=h.gjdgxs)

[***Introduction: 2***](#_heading=h.30j0zll)

[***Literature Review: 2***](#_heading=h.1fob9te)

[***Document Clustering 2***](#_heading=h.3znysh7)

[***Term Frequency-Inverse Document Frequency (TF-IDF) 3***](#_heading=h.2et92p0)

[***Cosine Similarity 3***](#_heading=h.tyjcwt)

[***Heap Data Structures 3***](#_heading=h.3dy6vkm)

[***Gap in the Literature 3***](#_heading=h.1t3h5sf)

[***Methodology: 4***](#_heading=h.4d34og8)

[***Preliminary Experiment 4***](#_heading=h.2s8eyo1)

[***Preliminary Experimentation and Large-scale Application 8***](#_heading=h.17dp8vu)

[***Adapting for Larger Datasets 8***](#_heading=h.3rdcrjn)

[***Experimentation with Larger Datasets 8***](#_heading=h.26in1rg)

[***Analysis and Results 9***](#_heading=h.lnxbz9)

[***Expected Output 11***](#_heading=h.35nkun2)

[***Limitation 11***](#_heading=h.1ksv4uv)

[***Conclusion 11***](#_heading=h.44sinio)

[***References 12***](#_heading=h.2jxsxqh)

# Abstract:

This paper presents an approach to cluster documents using term frequency-inverse document frequency (TF-IDF) to identify the most important terms in each document and then using cosine similarity implemented by heap data structures to sort the documents based on their cosine similarity to the root document.

# Introduction:

Document clustering is a crucial task in natural language processing and information retrieval. It involves grouping similar documents based on their content, which can help in many areas such as categorizing news articles, identifying spam emails, and identifying similar legal documents. However, clustering documents can be time-consuming for individuals since it requires human intuition to cluster and store essential documents for companies and big organizations.

Then my research question arises from here. Is there a more computationally efficient way to cluster large numbers of documents?

In this paper, I present an approach to document clustering that utilizes TF-IDF and cosine similarity implemented with heap data structures. My algorithm aims to guide individuals on how to organize documents inside a directory using heap data structures which has performance optimization since heaps have an average time complexity of O(log n) for insertion and deletion. Clustering algorithms help group similar documents based on their content, which can be achieved by calculating the similarity between each document using cosine similarity and TF-IDF.

# Literature Review:

# Document Clustering

Document clustering is a widely researched field and has many potential applications. For example, Steinbach, Karypis, and Kumar (2000) have used document clustering for trend detection in dynamic datasets[[1]](#footnote-1), while Dhillon and Modha (2001) applied it for text classification[[2]](#footnote-2). However, as data sets grow in size, traditional document clustering methods become computationally expensive, leading to the need for more efficient algorithms.

# Term Frequency-Inverse Document Frequency (TF-IDF)

The use of TF-IDF in document clustering has been demonstrated in several studies. Salton and Buckley (1988) presented TF-IDF as a way of weighting terms in documents, considering not only the frequency of a term in a particular document but also its rarity across all documents[[3]](#footnote-3). Ramos and others (2003) have shown that using TF-IDF can help reduce the dimensionality of the data and improve clustering performance[[4]](#footnote-4). Nevertheless, the issue of handling large volumes of data still remains a challenge.

# Cosine Similarity

Cosine similarity is a common method for calculating the similarity between documents. It is often used in combination with TF-IDF to create document vectors and measure their similarity[[5]](#footnote-5) (Singhal, 2001). This approach has proven successful in various applications, but when dealing with large data sets, the computational cost of calculating pairwise cosine similarities can be prohibitive.

# Heap Data Structures

Heap data structures have been used in numerous applications for their efficiency in handling large data sets[[6]](#footnote-6) (Cormen et al., 2009). They have been implemented in various fields, such as in task scheduling[[7]](#footnote-7) (Blelloch et al., 1991) and in developing efficient sorting algorithms[[8]](#footnote-8) (Williams, 1964). However, to the best of our knowledge, their use in document clustering, particularly in conjunction with TF-IDF and cosine similarity, has been relatively unexplored.

**Research Objective**

Testing on a small dataset allowed me to explore the algorithm's behavior and assess its feasibility on a limited scale.

While the results obtained from this small-scale study cannot be generalized, they provide insights into the algorithm's performance and potential areas for improvement.

To assess the generalizability and scalability of our proposed algorithm, I plan to conduct additional experiments on a larger dataset.

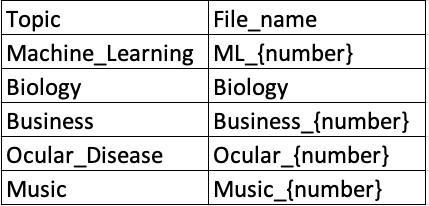
# Gap in the Literature

Although much work has been done in document clustering, there still exists a gap in research for efficiently handling large volumes of documents. The proposed approach in this study—combining TF-IDF, cosine similarity, and heap data structures—aims to fill this gap and contribute to the further advancement of efficient document clustering methods.

# Methodology:

# Preliminary Experiment

First, I conducted a preliminary experimental evaluation of my algorithm using a small dataset. There are a total of 18 documents inside a specified directory filename which function as class labels for the algorithm. Each class label has been selected after reading the document and was labeled in relation to the topic. (*Figure 1*)



*Figure 1. Description of document file and file names used to visualize the heap data structure*

The proposed approach uses TF-IDF to identify the most important terms in each document. Each document is represented as a vector in a high-dimensional space, with each unique term corresponding to a dimension. The method assigns a weight to each term in a document based on its frequency in the document and its frequency in the corpus. The time complexity of this step is typically O(n), where n is the total number of terms in the corpus. The approach then uses cosine similarity to calculate the similarity between documents based on the frequency of each term in the document. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space.[[9]](#footnote-9) The time complexity of this operation is usually O(n^2) in the worst case, as each document needs to be compared with every other document.

To improve the efficiency of the algorithm, we introduce the use of a heap data structure. Heaps are binary trees for which every parent node has a value less than or equal to any of its children. This property makes them useful for sorting and priority queue operations. The heap is used to maintain a collection of the documents that are most similar to the root document. Adding a document to the heap or removing the least similar document (when the heap is full) are both operations that have an average time complexity of O(log m), where m is the size of the heap. This is significantly faster than the naive approach of comparing each new document to all existing documents in the heap, which would have a time complexity of O(m).

In summary, the overall time complexity of the algorithm can be broken down into these major parts:

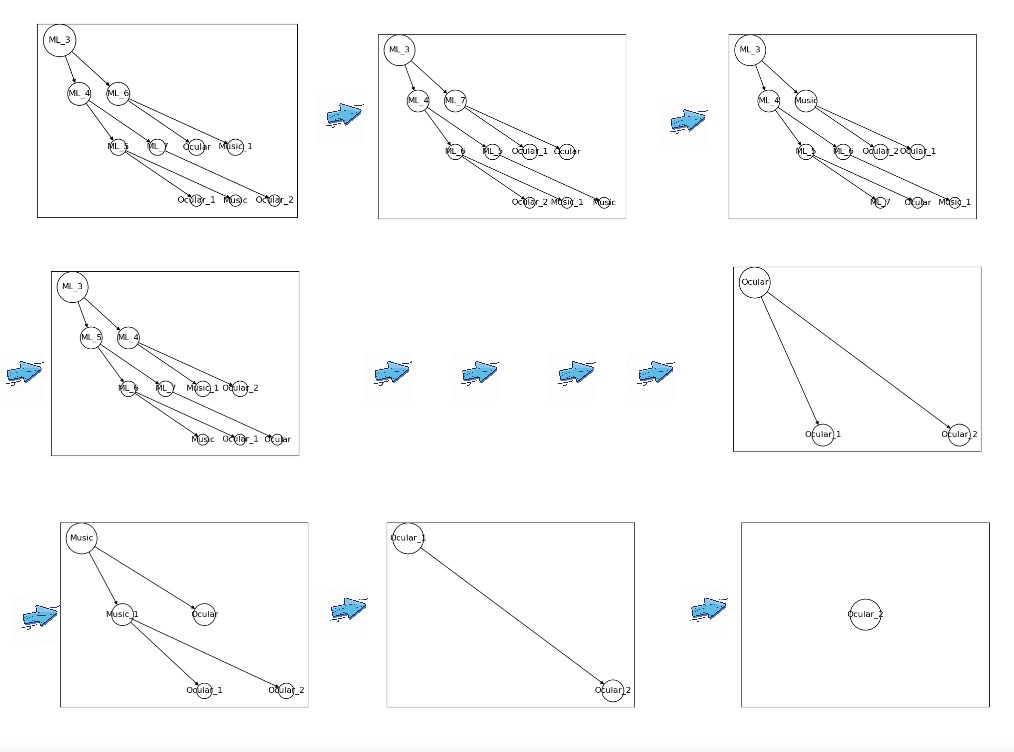
* Computing TF-IDF: O(n)
* Calculating cosine similarities: O(n^2)
* Managing the heap: O(log m) for each insertion or deletion

While the calculation of cosine similarities is potentially the most computationally expensive part of the algorithm due to its quadratic time complexity, the use of a heap data structure helps to reduce the cost of managing the most similar documents and makes the overall algorithm more efficient.

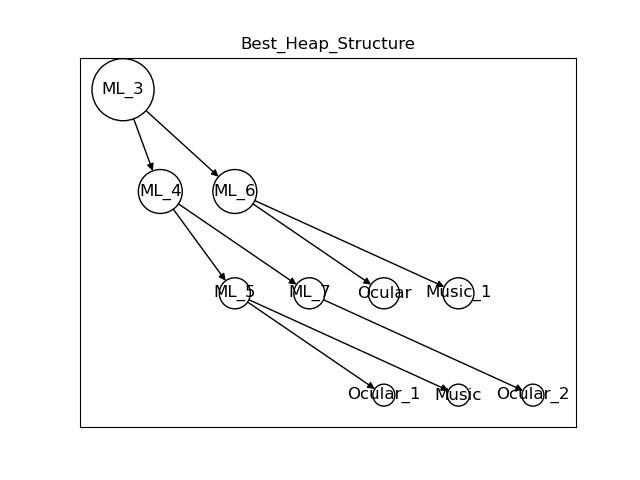
It's important to note that the actual performance of the algorithm will also depend on factors such as the specific implementation and the characteristics of the data (e.g., the number of unique terms, the distribution of terms across documents, and the size of the heap.

In the case of heap data structures, a higher threshold value would result in fewer documents being added to the heap data structure, while a lower threshold value would result in more documents being added[[10]](#footnote-10). The algorithm removes the document with the lowest cosine similarity if the heap is full. The algorithm then repeats the process with the next document, using the updated heap data structure. The algorithm uses the **networkx** and **matplotlib** libraries to draw a pyramid-like visualization of the heap data structure, based on preliminary experimentation with smaller datasets (Figure 2).

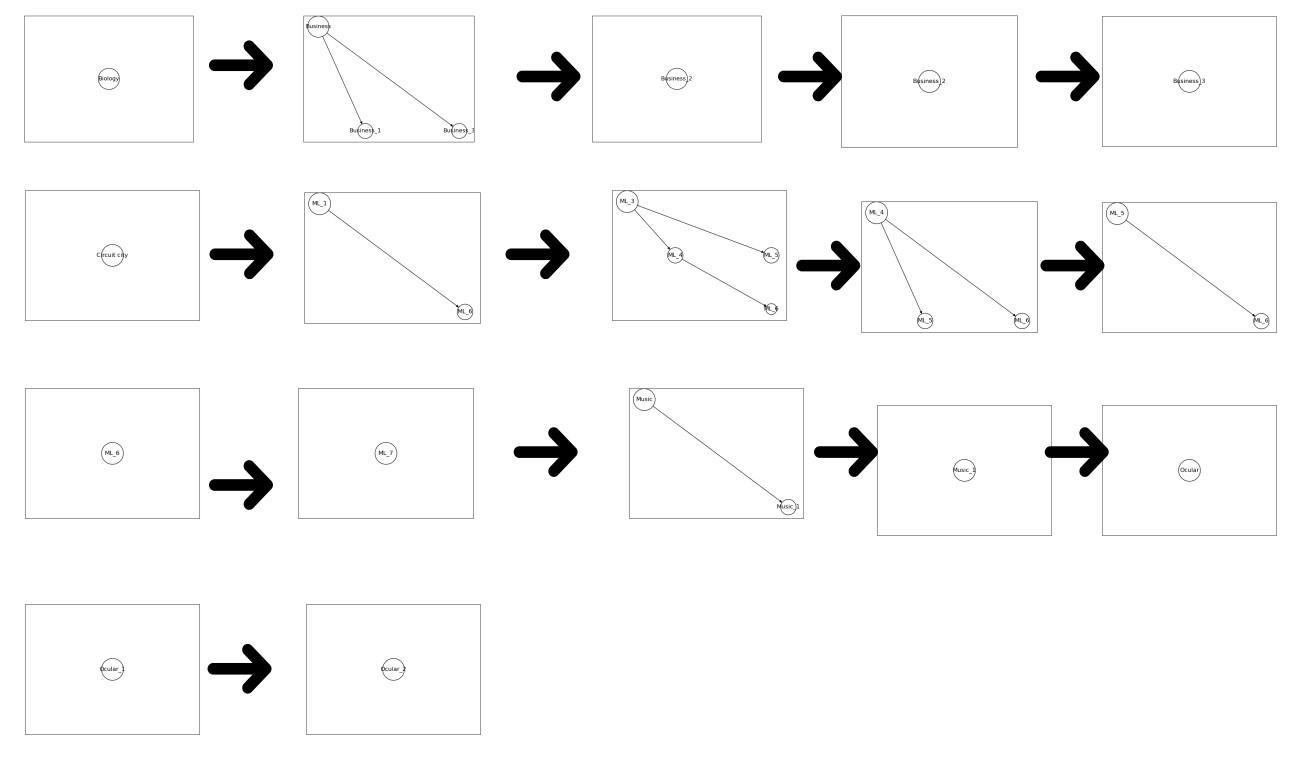
To choose the best heap structure from the iteration, the algorithm calculates the average cosine similarity of the documents in the heap. It keeps track of the heap with the highest average cosine similarity and returns it as the chosen heap. This process was also visualized using data from preliminary, smaller dataset experiments (Figure 3).



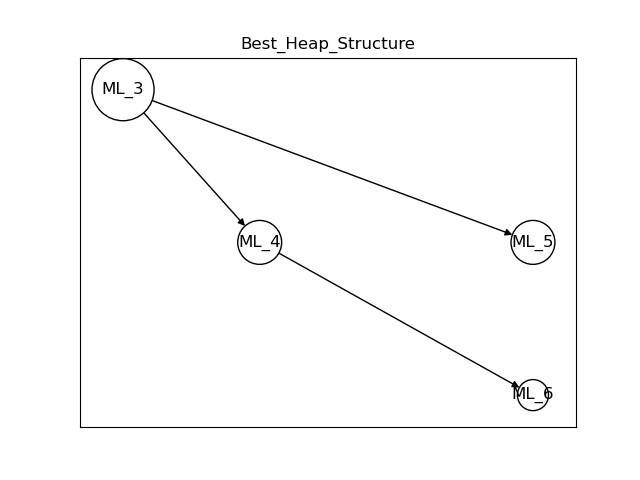
*Figure 2. Iterative visualization of heap data structure of 18 documents with threshold value = 0.01*



*Figure 3. Optimal heap data structure of 18 documents with threshold value = 0.01*



*Figure 4. Iterative visualization of heap data structure of 18 documents with threshold value = 0.1*



*Figure 5. Optimal heap data structure of 18 documents with threshold value = 0.1*

# Preliminary Experimentation and Large-scale Application

The methodology outlined in the preceding section was initially developed and tested using smaller datasets. This preliminary experimentation was crucial for debugging, fine-tuning, and validating the approach. Through this phase, we were able to understand the nuances of our algorithm and prepare it for application on a larger scale.

As the algorithm exhibited promising results and consistent performance with smaller datasets, we proceeded to apply it to larger datasets. This section details the process and considerations involved in this large-scale application.

# Adapting for Larger Datasets

When applying the algorithm to larger datasets, certain considerations needed to be made. The computational resources required scale with the size of the dataset. As such, we ensured that adequate computational power was available for processing these larger datasets, utilizing AWS cloud engine.

# Experimentation with Larger Datasets

The larger datasets used for the advanced stage of our research were drawn from diverse sources. Specifically, we focused on four distinct topics: baseball statistics, ocular disease detection with deep learning models, microservices, and cloud computing. For each topic, we collected 200 research papers from reputable academic databases, including PubMed and CORE. These research papers were carefully selected by me and my research partner, Jean, based on their relevance to the chosen topic.

In preparation for analysis, we implemented Latent Semantic Analysis (LSA) to extract seven keywords from each document (See Figure 6). This step was undertaken to avoid confusion that might arise from the originally random naming of the files. Furthermore, these keywords served two essential functions. Firstly, they provided us with a high-level understanding of the content within each document. Secondly, they proved invaluable when visualizing our heap structure later in the analysis process. We printed out the process of renaming the documents according to their topic to give a better understanding to our methodology.

The terminal output was as follows:

Topic 1: ability,productivity,married,experience,marriage,wages,player

Topic 1: advertising,baseball,game,screen,study,field,stadium

Topic 1: age,break,barry,performance,time,two,player

Topic 1: analysis,baseball,statistical,data,statistics,season,historical

Topic 1: anticipation,batting,occlusion,baseball,statistics,based,information

*Figure 6. Sample output of topic keywords from each document*

# Analysis and Results

To further aid our exploration and interpretation of the larger dataset, we employed the k-means clustering algorithm and generated word cloud visualizations for each cluster. K means algorithm that was implemented for this experiment identifies the top 10 terms for each cluster and are identified and fed into the WordCloud module, which generates a word cloud image for each cluster (See Figure 7).

A close-up of words

Description automatically generated with low confidenceA close-up of words

Description automatically generated with low confidence

Add more figures of wordcloud fitted inside a box

*Figure 7. Word cloud sample of a cluster*

Moreover, we sought to evaluate and compare the performance of different heap structures generated during the document clustering process. The Silhouette Coefficient, a well-known metric for assessing clustering quality, was utilized for this task.

The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. For the context of our study, 'a' is interpreted as the mean similarity of a document to the other documents in its own heap (i.e., its cluster), and 'b' is defined as the mean of the maximum similarity of the document to documents in other heaps.

Our methodology mirrored the steps we initially used in a preliminary experiment with a smaller dataset, albeit with some modifications to accommodate the large volume of documents in this study. We developed the function get\_heap\_lst to produce a list of all heap structures with root document. Given the considerable number of documents, it was necessary to impose restrictions on what would be included in the best\_heap\_lst. This list would later be instrumental in comparing and selecting the most efficient heap data structure.

The function then iteratively creates heaps of documents based on cosine similarity, using a predefined threshold to decide whether a document should be added to the heap. Only heaps with more than two documents are added to the final list of "best heaps.".

For each document, the function then maps it to its corresponding heap index, creating a dictionary. The document names are updated to include only those documents that have been successfully mapped to a heap. Using this updated dictionary, we generate labels indicating the heap number (cluster) each document belongs to.

Finally, the function computes and returns a list of Silhouette Scores for each heap. To ensure the correctness of the Silhouette Score calculation, the score is only computed for heaps (clusters) that contain more than one document.

This evaluation process was critical in understanding how effectively our heap-based clustering algorithm groups similar documents together, and helped us in identifying the heap structures that yielded the highest quality clusters. The Silhouette Scores computed provided a quantifiable metric to compare and contrast the performance of different heaps, thus guiding us in fine-tuning our clustering algorithm for optimal performance.

The experimentation with larger datasets allowed us to evaluate the robustness and scalability of our algorithm. We recorded the algorithm's performance metrics, such as processing time and resource usage, and compared them to its performance with smaller datasets. The results of these tests (Fig 6.) add terminal output of each heap’s silhouette score affirmed our approach's effectiveness and scalability

# Limitation

There are a few limitations to my approach. Firstly, it is difficult to quantify the comparison results in terms of accuracy and computation time between my proposed clustering algorithm and other clustering algorithms, such as K-means. Secondly, there is no perfect solution for finding the optimal threshold for cosine similarity, as it depends on the specific use case and domain structure.

# Conclusion

# References

* Steinbach, M., Karypis, G., & Kumar, V. (2000). A comparison of document clustering techniques. In KDD workshop on text mining, Boston (Vol. 400, No. 1, pp. 525-526).
* Dhillon, I. S., & Modha, D. S. (2001). Concept decompositions for large sparse text data using clustering. Machine learning, 42(1-2), 143-175.
* Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information processing & management, 24(5), 513-523.
* Ramos, J., et al. (2003). Using TF-IDF to Determine Word Relevance in Document Queries. In Proceedings of the First Instructional Conference on Machine Learning.
* Singhal, A. (2001). Modern information retrieval: A brief overview. IEEE Data Eng. Bull., 24(4), 35-43.
* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.
* Blelloch, G. E., Leiserson, C. E., Maggs, B. M., Plaxton, C. G., Smith, S. J., & Zagha, M. (1991). A comparison of sorting algorithms for the connection machine CM-2. In Proceedings of the third annual ACM symposium on Parallel algorithms and architectures - SPAA '91 (pp. 3-16). ACM Press. <https://doi.org/10.1145/113251.113252>
* Williams, J. W. J. (1964). Algorithm 232 - Heapsort. Communications of the ACM, 7(6), 347-348. <https://doi.org/10.1145/512274.512284>
* metric - Cosine similarity vs The Levenshtein distance - Data Science .... https://datascience.stackexchange.com/questions/63325/cosine-similarity-vs-the-levenshtein-distance
* Zhang, Xiaodan & Hu, Xiaohua & Zhou, Xiaohua. (2008). A comparative evaluation of different link types on enhancing document clustering. 555-562. 10.1145/1390334.1390429.

1. Steinbach, M., Karypis, G., & Kumar, V. (2000). A comparison of document clustering techniques. In KDD workshop on text mining, Boston (Vol. 400, No. 1, pp. 525-526). [↑](#footnote-ref-1)
2. Dhillon, I. S., & Modha, D. S. (2001). Concept decompositions for large sparse text data using clustering. Machine learning, 42(1-2), 143-175. [↑](#footnote-ref-2)
3. Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information processing & management, 24(5), 513-523. [↑](#footnote-ref-3)
4. Ramos, J., et al. (2003). Using TF-IDF to Determine Word Relevance in Document Queries. In Proceedings of the First Instructional Conference on Machine Learning. [↑](#footnote-ref-4)
5. Singhal, A. (2001). Modern information retrieval: A brief overview. IEEE Data Eng. Bull., 24(4), 35-43. [↑](#footnote-ref-5)
6. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press. [↑](#footnote-ref-6)
7. Blelloch, G. E., Leiserson, C. E., Maggs, B. M., Plaxton, C. G., Smith, S. J., & Zagha, M. (1991). A comparison of sorting algorithms for the connection machine CM-2. In Proceedings of the third annual ACM symposium on Parallel algorithms and architectures - SPAA '91 (pp. 3-16). ACM Press. https://doi.org/10.1145/113251.113252 [↑](#footnote-ref-7)
8. Williams, J. W. J. (1964). Algorithm 232 - Heapsort. Communications of the ACM, 7(6), 347-348. https://doi.org/10.1145/512274.512284 [↑](#footnote-ref-8)
9. metric - Cosine similarity vs The Levenshtein distance - Data Science ....

   h8ps://datascience.stackexchange.com/quesAons/63325/cosine-similarity-vs-the-levenshtein-distance [↑](#footnote-ref-9)
10. Zhang, Xiaodan & Hu, Xiaohua & Zhou, Xiaohua. (2008). A comparative evaluation of different link types on enhancing document clustering. 555-562. 10.1145/1390334.1390429. [↑](#footnote-ref-10)