Optimizing Document Clustering through Correlation-Driven Cluster Formation

Suraj Kashyap[1] and Uttam Mahata[1]

[1] Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur, Howrah, India

Abstract

Document clustering is a vital component in organizing extensive datasets for efficient information retrieval and analysis. This paper presents an innovative method for enhancing clustering algorithms by dynamically adjusting parameters based on correlation thresholds. Utilizing Natural Language Processing for textual data preprocessing, the study evaluates the approach using established clustering metrics. Results demonstrate the method's adaptability to diverse datasets, yielding notable improvements in clustering accuracy and efficiency. The identified optimal threshold values offer valuable insights for parameter selection, making this approach a significant contribution to clustering optimization. The findings hold promise for advancing the practicality of clustering algorithms in various data-driven applications, particularly in domains like data science and machine learning.

Keywords

Information Retrieval, Threshold Value, Spearman Correlation Coefficient, Jaccard Coefficient, Cluster Formation, Error Calculation, Elbow Method, Comparative Analysis, K-Means Algorithm, Affinity Propagation, Gaussian Mixture Model, Agglomerative Clustering, Silhouette Score, Davies-Bouldin Index, Calinski Harabasz Score, Adjusted Rand Score, Normalized Mutual Information Score

1 Introduction

In the ever-expanding digital content landscape, effective document clustering plays a pivotal role in organizing extensive datasets for efficient information retrieval and analysis. This research paper presents an innovative approach to document clustering optimization through correlation-driven techniques. The methodology integrates a sophisticated preprocessing pipeline, encompassing tokenization, lemmatization, and TF-IDF vectorization, laying the foundation for a dynamic optimization process.

Document clustering is a crucial task in information retrieval and text mining. In this literature review, we explore various optimization-driven and descriptive document clustering approaches proposed by researchers. Jensi and Jiji [5] conducted a survey on optimization approaches to text document clustering. They compared the performance of techniques such as K-means, Fuzzy C-means, and Genetic Algorithm on different datasets, highlighting the dependency on both the dataset and the optimization technique used. Kayest and Jain [6] proposed an optimization-driven cluster-based indexing and matching approach for document retrieval. Their method, outperforming traditional indexing techniques, utilized clustering to group similar documents and an indexing technique for efficient retrieval. Brockmeier et al. [3] introduced a self-tuned descriptive document clustering approach using a predictive network. By employing a neural network to predict the optimal number of clusters, they demonstrated improved performance compared to traditional clustering algorithms. Tolner et al. [8] proposed a clustering approach based on LDA topic modeling for business organizations. They used a topic modeling algorithm to extract topics from documents, followed by a clustering algorithm for grouping. This approach exhibited superior performance compared to traditional methods. Alharbi et al. [1] enhanced topic clustering for Arabic security news using k-means and topic modeling. Their method showcased improved performance compared to traditional clustering algorithms. Wahid and

Hassini [9] conducted a literature review on correlation clustering, providing a cross-disciplinary taxonomy and analyzing developments from 1992 to 2020. The authors presented mathematical formulations and solution approaches, identified gaps, and proposed future research directions.

Our work makes the following contributions:

- Dataset Preparation
- Data Preprocessing
- Cluster Formation
- Identification of Optimum Threshold Value

In this paper, we propose a novel approach to topic modeling and clustering, based on the answers to the questions collected from arbitrary four different topics. We preprocess the data using a series of steps: Tokenization, Lemmatization, Filtering stopwords and punctuation, TF-IDF Vectorization. Then we calculate the Spearman correlation coefficient between each pair of questions. Formation of clusters is done by setting a threshold value and creating fields with values greater than the average and the threshold. Jaccard coefficient is used to measure the similarity between the fields and merge them based on the maximum value. We label the fields cluster-wise, assign a separate label to fields not belonging to any cluster, and verify the relational significance of fields present in multiple clusters. We vary the threshold value and calculate the error and rate of change of error to find the optimum threshold value for each dataset. We compare our algorithm with well-known clustering algorithms using various evaluation indices. We interpret and discuss the results in the context of our research objectives and existing literature. The ensuing sections will delve deeper into the methodology, experimental setups, and comparative analyses, positioning our approach within the broader context of advancements in clustering algorithms for various data types.

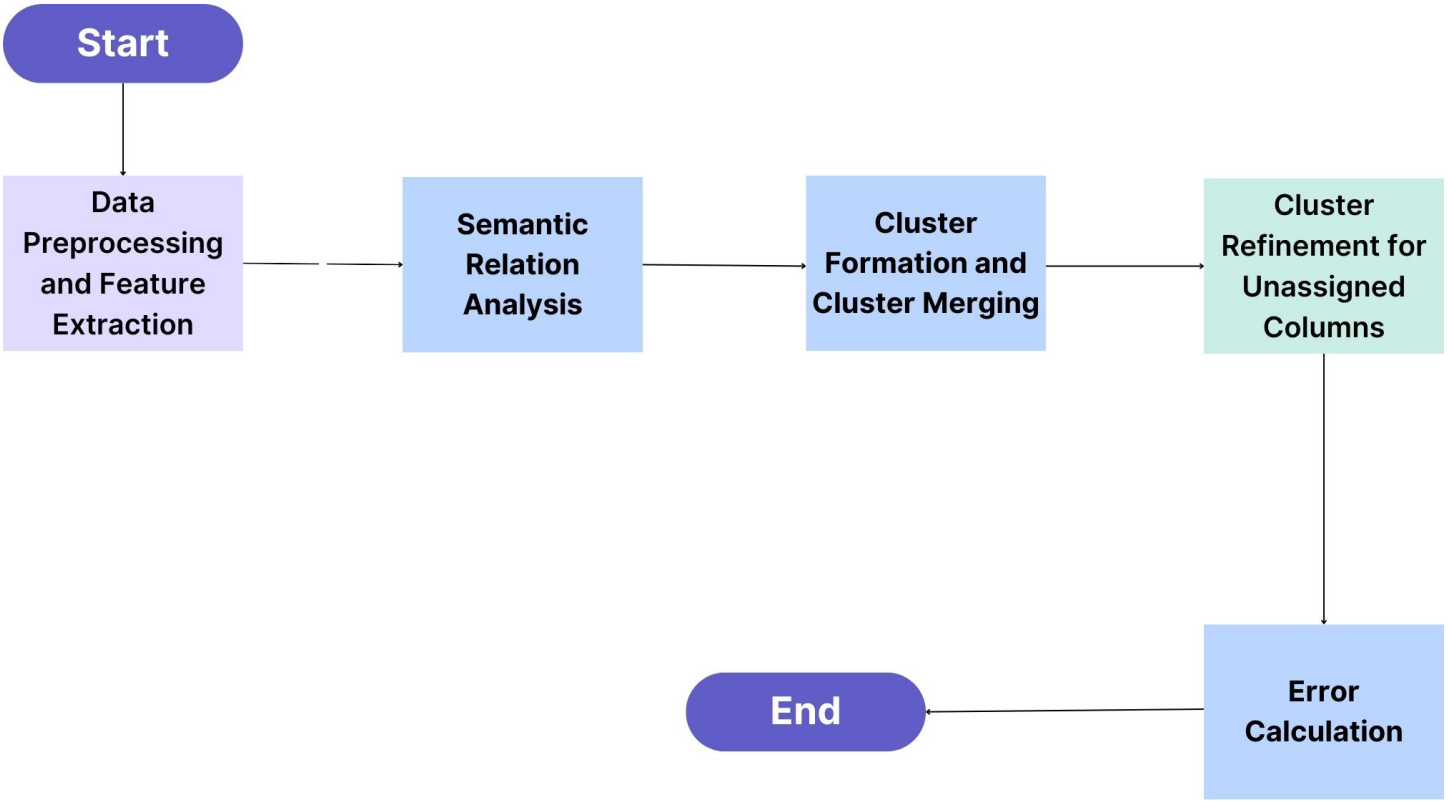

Figure 1: Flow Diagram for Steps

2 Methodology

The Threshold-based Correlation Clustering (TBCC) methodology orchestrates a systematic process for optimizing clusters using TF-IDF vectors' correlation matrix. Beginning with data preprocessing involving tokenization and lemmatization, the methodology ensures a standardized and cleaned dataset. TF-IDF vectors are then generated, providing the foundation for Spearman correlation-based correlation matrix calculation. The clustering process identifies cohesive semantic groups by considering above-average correlations. The Jaccard coefficient is employed to quantify overlap between clusters, guiding the iterative optimization through threshold-based merging. Convergence is reached when no pair of clusters exhibits a maximum correlation above a specified threshold. To enhance clustering quality, an iterative refinement process addresses unassigned columns. In essence, TBCC systematically refines clusters, emphasizing semantic coherence and distinctiveness within the dataset, making it a robust approach for uncovering meaningful relationships in textual data.

2.1 Data Preprocessing and Feature Extraction

The preprocessing phase involves eliminating stop words and applying stemming techniques to refine the text database. Stop words, such as 'a,' 'an,' 'in,' 'the,' are excluded during searches, streamlining the retrieval of meaningful results. Simultaneously, stemming reduces words to their root form, enhancing comparability during observations. The resulting dictionary words serve as the foundation for subsequent feature extraction, ensuring a more precise and optimized text dataset for advanced analyses within the TBCC methodology.

Feature extraction is pivotal in transforming textual data into quantifiable features for meaningful analysis. It involves generating TF-IDF (Term Frequency-Inverse Document Frequency) vectors, a numerical representation that gauges the importance of each term within a document relative to the entire dataset [7]. The weight assigned by TF-IDF amplifies with a term's prevalence in a document, underscoring its significance, and diminishes for ubiquitous terms. The Inverse Document Frequency calculates the importance of rarely occurring words. The TF-IDF is expressed as:

$$\text{TF-IDF}(A, B, E) = S(A, B) \times T(A, E)$$

where $S(A, B)$ is the Term Frequency (TF), $T(A, E)$ is the Inverse Document Frequency (IDF), A denotes words, B denotes each document, and E denotes the collection of documents. The IDF formula is:

$$\text{IDF}(A, E) = \log\left(\frac{N}{1 + \text{df}(A, E)}\right)$$

Here, A is the term, E is the document collection, N is the total number of documents in E, and $\text{df}(A, E)$ is the document frequency of term A in E.

Pseudo-Code

```
function preprocess_documents(input_folder, output_file)
    # For each file in the input folder read the file content and add it to the documents
    documents = []
    for file in input_folder
        documents.append(read_file(file))

    # Filter out stopwords, punctuation, and irrelevant words from each document
    filtered_documents = []
    for document in documents
        for word in document
            doc=[]
            if word not a stopwords, punctuation, length(word) == 1, not isalpha()
                # Lemmatize each word in each document
                word = lemmatize(word)
                doc.append(word)
            # Convert back to a string
            doc = join_words(doc)
            filtered_documents.append(doc)

    # Create a TF-IDF vectorizer and fit it to the preprocessed documents
    tfidf_matrix = create_tfidf_vectorizer(preprocessed_documents)
    feature_names = get_feature_names(tfidf_vectorizer)
    tfidf_df = create_dataframe(tfidf_matrix, feature_names)

    # Save the dataframe to a CSV file
    save_dataframe(tfidf_df, output_file)
```

2.2 Semantic Relationship Analysis

The procedure for semantic relationship analysis commences with the calculation of the correlation matrix (C) derived from TF-IDF vectors. Spearman's Rank Correlation Coefficient (ρ) is employed to robustly measure monotonic relationships between words, ensuring a comprehensive understanding of semantic associations.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

where d_i is the difference between the ranks of corresponding TF-IDF values for the i-th pair of words, and n is the number of pairs.

This step guarantees that the correlation matrix captures both linear and potential nonlinear dependencies, enriching the semantic insights [2].

Pseudo-Code

```
# Calculate the Spearman correlation coefficient between each pair of questions
# Generate a symmetric matrix of order 200x200
correlation_matrix = [[0 for i in range(N] for j in range(N)] # create an empty matrix of zeros
for i in range(N)
  for j in range(N)
    # calculate the correlation coefficient
    correlation_matrix[i][j] = calc_spearman(tfidf_matrix[i], tfidf_matrix[j])
```

2.3 Cluster Formation and Cluster Merging

The Cluster Formation step plays a pivotal role in capturing semantic coherence among words by creating groups with above-average correlations. The process begins with a row-wise calculation, evaluating the average correlation (Average Corr_k) for each word. This provides a measure of each word's overall association with others in the dataset. Clusters are then formed by selecting words with individual column-wise average correlation values ($Corr_{ki}$) surpassing the global average. The criterion for inclusion in a cluster is that the individual correlation values must exceed the specified threshold (T).

$$Cluster_k = \{word_i \mid Corr_{ki} > \text{Average Corr}_k > T\}$$

This clustering strategy emphasizes grouping words with stronger-than-average semantic relationships, laying the foundation for subsequent optimization steps in the TBCC methodology.

Following Cluster Formation, the methodology proceeds to calculate the Jaccard Coefficient between pairs of clusters. The Jaccard Coefficient quantifies the degree of overlap between clusters and serves as a meaningful metric for assessing the distinctiveness and coherence of the formed semantic clusters. The Jaccard Coefficient between two clusters A and B is computed as:

$$\text{Jaccard Coefficient}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

This measure provides valuable insights into the similarity between clusters, contributing to the evaluation of the overall quality and cohesion of the formed semantic groups. The evaluation is informed by multiple sources, including the research conducted by Irani et al. [4], which discussed clustering techniques and the similarity measures used in clustering. The TBCC methodology iteratively optimizes clusters by merging those with Jaccard coefficients above the specified threshold T. This iterative process ensures the formation of meaningful and distinct clusters. The merging criterion is defined as:

$$\text{Merge Cluster } A \text{ and Cluster } B \text{ if Jaccard Coefficient}(A, B) \geq T$$

This approach systematically refines clusters, producing semantically cohesive clusters representing distinct concepts or themes within the dataset.

The optimization process within TBCC continues until convergence, ensuring that no pair of clusters exhibits a maximum correlation above the specified threshold T. The convergence criteria can be expressed as:

$$\max(C_{ij}) < T$$

This condition signifies that the optimization process converges when the maximum correlation between any pair of clusters falls below the threshold. To refine the clustering results, an iterative procedure assesses columns initially unassigned to any clusters. The algorithm identifies clusters with a single-column representation and evaluates the relationships between unassigned columns and those clusters. The refinement process contributes to the overall quality of clustering by addressing any unassigned or outlier columns within the dataset. These steps in the TBCC methodology collectively contribute to the systematic optimization of clusters, producing semantically cohesive and distinct groupings within the textual data.

Pseudo-Code

```
# Define the number of questions and topics
N = ( Total number of questions )

# Import modules
import numpy as np
import pandas as pd

# Form clusters by setting a threshold value (T)
# Create a set of fields with values greter than average and T
# Apply Jaccard coefficient to create another 200x200 matrix
# Identify the max value(>T) in the matrix and merge the corresponding (i, j) pair into a single field
# Repeat this process until no further merges are possible
T = 0.2                                              # set the initial threshold value
fields = []                                          # create an empty list to store the fields
for i in range(N)                                    # loop over each row of the correlation matrix
  average = np.mean(correlation_matrix[i])           # calculate the average value of the row
  field = []                                         # create an empty list to store the field
  for j in range(N)                                  # loop over each column of the correlation matrix
    if correlation_matrix[i][j] > (average and T)    # if the value is greater than the average and T
      field.append(j)                                # append the index of the value to the field
  fields.append(field)                               # append the field to the list of fields

jaccard_matrix = np.zeros((N, N))                    # Create a Jaccard matrix from the fields
for i in range(N)
  for j in range(N)
     # calculate the Jaccard coefficient and store it in the matrix
    numerator = len(set(fields[i]).intersection(set(fields[j])))
    denominator = len(set(fields[i]).union(set(fields[j])))
    jaccard_matrix[i][j] = numerator / denominator

# Merge the fields based on the Jaccard matrix
merged_fields = fields.copy()                         # create a copy of the fields list
while True                                            # loop until no further merges are possible
  max_value = np.max(jaccard_matrix)                  # find the maximum value in the Jaccard matrix
  if max_value < T: break                             # check if the maximum value is less than to T
  else
    max_index = np.where(jaccard_matrix == max_value) # find the index of the maximum value
    i = max_index[0][0]                               # get the row index
    j = max_index[1][0]                               # get the column index
    merged_field = list(set(merged_fields[i]).union(set(merged_fields[j]))) # merge fields at i and j
    merged_fields[i] = merged_field                   # update the field at i
    merged_fields[j] = []                             # empty the field at j
    jaccard_matrix[i][j] = 0                          # set the Jaccard value at i and j to zero

merged_fields = [field for field in merged_fields if field] # Remove the empty fields
```

2.4 Cluster Refinement for Unassigned Columns

To refine the clustering results, an iterative procedure assesses columns initially unassigned to any clusters. The algorithm identifies clusters with a single column representation and evaluates the relationships between unassigned columns and those clusters.

Refinement Criteria: For each unassigned column, it assesses its relationship with columns in clusters with a single representation. A refinement criterion stipulates that if at least 75 percent of columns within the same original cluster support

the assignment of an unassigned column, it is reassigned to that cluster.

Assignment Process: If the criteria are met, the unassigned column is reassigned to the identified cluster. If the criteria are not met, the column is treated as a singleton cluster.

Pseudo-Code

```
function get_labels(clusters, original_clusters)
  # Initialize a list of labels with -1 for each data point
  labels = [-1 for i in range(self.data_length)]
  for c, ln in enumerate(clusters)           # Assign the labels by the cluster index
    for v in ln
      if labels[v] == -1
        labels[v] = c

  # Count the frequency of each label
  label_count = count(labels)
  for lbl in range(self.data_length)
    if label_count[labels[lbl]] == 1         # If the label count is one
      # Get the frequency of the labels in the original clusters of the label
      d = count([labels[k] for k in original_clusters[lbl] if k != lbl])
      s = sum(d.values())                    # Sum the values of the dictionary
      for k, i in d.items()                  # Loop through the keys and values of the dictionary
        # If the ratio of the value to the sum is greater than or equal to 0.75
        if round(i / s, 2) >= 0.75
          labels[lbl] = k                    # Assign the key as the label and break the loop
          break
      else
        labels[lbl] = -1                     # Assign -1 as the label

  # Count the frequency of each label
  label_count = count(labels)
  c = 0 # Initialize a counter for the cluster index
  d = {} # Initialize an empty dictionary
  for k in label_count.keys()
    if k != -1               # If the key is not -1
      d[k] = c               # Assign the counter as the value of the key and
      c = c + 1              # Increment the counter by one

  # Loop through the labels
  for i in range(self.data_length)
    # If the label is -1, assign the counter as the label and increment the counter by one
    # Else, assign the value of the label from the dictionary as the label
    labels[i] = c if labels[i] == -1 else d[labels[i]]

  return labels
```

3 Experimental Setup

We employed a dataset consisting of 200 questions across four topics: Biotechnology, Database Management System (DBMS), Network and Networking, and Climate Change. Each topic comprised 50 questions. The selection of topics was arbitrary, ensuring no specific influence on the clustering algorithm's functionality. The initial phase involved transforming raw text data for clustering, including tokenization, lemmatization, stopword and punctuation filtering, and TF-IDF vectorization.

This process resulted in numerical vectors, forming the foundation for evaluating the clustering algorithm's performance and comparing it against established methods.

The experiments were conducted using the TBCC methodology, as detailed in the previous methodology section. The implementation utilized the following software and tools: Python, Scikit-learn, NumPy, Pandas, NLTK for tokenization and lemmatization. The implementation followed the steps outlined in the TBCC methodology, encompassing data preprocessing, TF-IDF vectorization, semantic relationship analysis, cluster formation, evaluation of cluster overlap using the Jaccard Coefficient, and threshold-based cluster merging.

In our experiment, the focus was on determining the optimum threshold value for our custom clustering algorithm, a critical factor influencing the strength of semantic correlations between words and, consequently, the formation of clusters. The **threshold values** were systematically varied from 0.2 to 0.3 in increments of 0.01 for each iteration. For each threshold value, we calculated the corresponding number of clusters and the associated error.

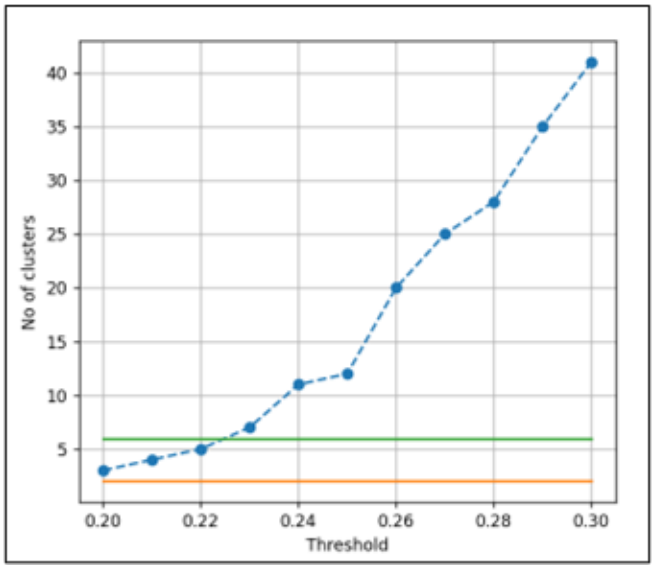

Figure 2: No of Clusters vs Threshold value

The process involved the following steps:

Centroid Calculation: In clustering algorithms, particularly within our methodology, the computation of centroids plays a crucial role in unraveling and characterizing the inherent qualities of clusters. The centroid (Centroid_k) for a specific cluster (Cluster_k) is calculated using the equation:

$$\text{Centroid}_k = \frac{\sum_{i \in \text{Cluster}_k} \text{TF-IDF}_i}{\text{Number of Documents in Cluster}_k}$$

Identification of Optimum Threshold: The elbow method, a widely employed technique, entails plotting the rate of change in error against various threshold values. The optimal threshold is discerned at the juncture where the rate of error reduction experiences a significant slowdown, resembling the characteristic bend of an elbow in the plot. This threshold value strikes a balance between cluster coherence (minimizing error) and meaningfulness, ensuring a reasonable number of clusters. The assessment of clustering quality hinges on accurate error calculation. Our methodology adopts the Root Mean Square (RMS) difference between formed clusters and their corresponding cluster centroids. The RMS calculation for a document-cluster pair is given by:

$$\text{RMS(Document, Cluster)} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

where n is the dimension of the vectors, x_i represents individual vector components, and $\bar{x}$ denotes the mean. Cluster-wise

error (E_{cluster}) is the average RMS value across all documents in a cluster:

$$E_{\text{cluster}} = \frac{\sum_{j=1}^{m} \text{RMS}(\text{Document}_j, \text{Cluster})}{m}$$

where m is the number of documents in the cluster. Global error (E_{global}) is the average of cluster-wise errors across all clusters:

$$E_{\text{global}} = \frac{\sum_{k=1}^{K} E_{\text{cluster}_k}}{K}$$

where K is the total number of clusters. This comprehensive error assessment framework ensures a nuanced evaluation of clustering effectiveness.

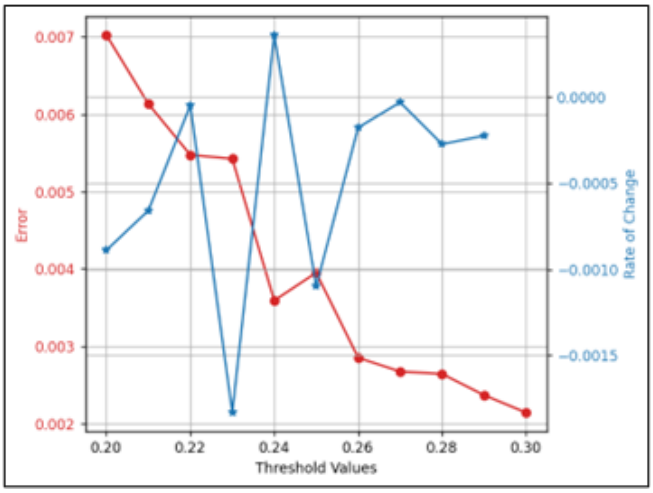

Figure 3: Error vs Threshold value

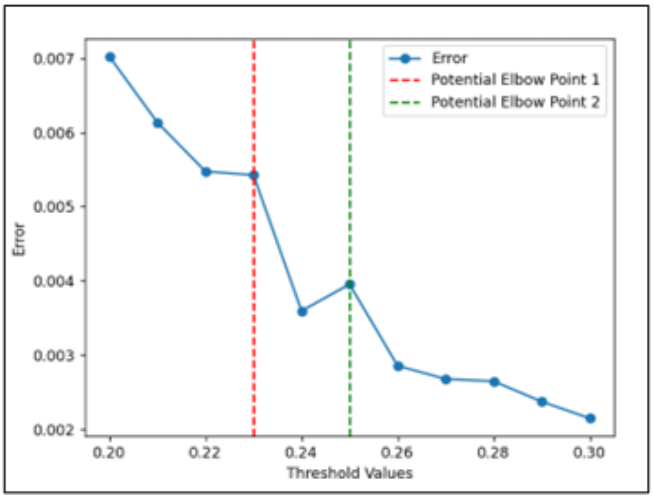

Figure 4: Identification of Optimum Threshold value

The custom clustering algorithm demonstrates efficient performance, with clustering, centroid and error calculation taking approximately 3-4 seconds.

4 Results and Discussions

The results obtained from the experiments provide valuable insights into the performance of the proposed Custom Clustering Algorithm compared to traditional clustering approaches. After determining the optimum threshold value, we set this value for subsequent analyses, ensuring consistent and comparable results. The fixed threshold value provides stability to the clustering process, allowing for a more focused evaluation of the algorithm's performance. The custom algorithm's performance was assessed against other clustering algorithms at the optimum number of clusters, determined by the identified threshold value. The custom algorithm's results were compared with other algorithms under the constraint of the optimum number of clusters and original number of clusters. The number of clusters was set to four, reflecting the original categorization based on the four chosen topics (Biotechnology, DBMS, Network and Networking, and Climate Change). This parameterized strategy allowed us to not only optimize the algorithm's performance under its intrinsic settings but also evaluate its adaptability to specific scenarios, contributing to a comprehensive understanding of its capabilities.

4.1 Comparative Analysis

In this paper, we have proposed a novel approach to topic modeling and clustering, based on the answers to the questions collected from four different topics. We have compared our algorithm with well-known clustering algorithms, such as KMeans, Affinity Propagation, Gaussian Mixture, and Agglomerative Clustering, using various evaluation indices, such as Davies-Bouldin Index, Silhouette Score, Calinski Harabasz Score, Adjusted Rand Score, and Normalized Mutual Info Score.

- Comparative analysis when number of clusters is set to numbers of clusters corresponding to the optimum the Threshold value (T=25):

  Davies-Bouldin Index: This index measures the average similarity between clusters, where lower values indicate better clustering. Based on this index, our custom clustering algorithm (T=0.25) has the best performance, followed by Affinity Propagation and our custom clustering algorithm (T=0.23). KMeans, Gaussian Mixture, and Agglomerative Clustering have relatively poor performance, indicating that the clusters are not well-separated or compact.
  Silhouette Score: This score measures how well each point fits in its cluster, where higher values indicate better clustering. Based on this score, Affinity Propagation has the best performance, followed by your custom clustering algorithms (T=0.23 and T=0.25). Gaussian Mixture has the worst performance, indicating that the points are assigned to wrong clusters or the clusters are overlapping.
  Calinski Harabasz Score: This score measures the ratio of between-cluster variance to within-cluster variance, where higher values indicate better clustering. Based on this score, Affinity Propagation has the best performance, followed by your custom clustering algorithm (T=0.25) and KMeans. Your custom clustering algorithm (T=0.23), Gaussian Mixture, and Agglomerative Clustering have relatively low performance, indicating that the clusters are not well-defined or distinct.
  Adjusted Rand Score: This score measures the similarity between two clustering, where higher values indicate better clustering. This score can be used to compare your algorithm with the ground truth labels. Based on this score, our custom clustering algorithms (T=0.23 and T=0.25) and Affinity Propagation have the best performance, indicating that they are consistent with the ground truth labels. KMeans, Gaussian Mixture, and Agglomerative Clustering have relatively low performance, indicating that they are not consistent with the ground truth labels.
  Normalized Mutual Info Score: This score measures the mutual information between two clustering, where higher values indicate better clustering. This score can also be used to compare our algorithm with the ground truth labels. Based on this score, our custom clustering algorithms (T=0.23 and T=0.25) and Affinity Propagation have the best performance, indicating that they share a lot of information with the ground truth labels. KMeans, Gaussian Mixture, and Agglomerative Clustering have relatively low performance, indicating that they share little information with the ground truth labels.
  Based on the above analysis, We can conclude that our custom clustering algorithm (T=0.25) has the best overall performance, as it has the lowest Davies-Bouldin Index, the second highest Silhouette Score, the second highest Calinski Harabasz Score, the highest Adjusted Rand Score, and the highest Normalized Mutual Info Score. Affinity Propagation is also a good alternative, as it has the highest Silhouette Score and Calinski Harabasz Score, and the second highest Adjusted Rand Score and Normalized Mutual Info Score. However, it has a higher Davies-Bouldin Index than your algorithm, which means that it may create more clusters than necessary. KMeans, Gaussian Mixture, and Agglomerative Clustering have relatively poor performance, as they have high Davies-Bouldin Index, low Silhouette Score, low Adjusted Rand Score, and low Normalized Mutual Info Score. This means that they may create overlapping, noisy, or inconsistent clusters that do not match the ground truth labels.

Model	Davies-Bouldin Index	Silhouette Score	Calinski Harabasz Score	Adjusted Rand Score	Normalized Mutual Info Score
Custom Clustering Algorithm (0.23)	1.0755	0.2502	63.9523	0.8651	0.8684
Custom Clustering Algorithm (0.25)	0.8524	0.2445	122.0241	0.8721	0.8694
KMeans	2.2553	0.1106	124.9375	0.6236	0.7595
Affinity Propagation	1.3178	0.2664	297.9176	0.8666	0.8675
Gaussian Mixture	2.8988	0.0506	122.9254	0.4891	0.6936
Agglomerative Clustering	2.0598	0.1198	128.7157	0.5726	0.7294

Table 1: Optimum Number of Clusters

- Comparative analysis when number of clusters is set to numbers of clusters corresponding to the no of topics (N=4): Our experimental results show that our algorithm can create well-separated, compact, and consistent clusters that match the ground truth labels. Our algorithm also outperforms the other algorithms in terms of Davies-Bouldin Index, Adjusted Rand Score, and Normalized Mutual Info Score, indicating that it can create fewer, more coherent, and more informative clusters. Our algorithm has a slightly lower performance in terms of Silhouette Score and Calinski Harabasz Score, indicating that it can create less distinct clusters. However, this can be improved by tuning the threshold value and the similarity measures. We conclude that our novel approach to topic modeling and clustering is effective and efficient, and can be useful for applications such as content recommendation, summarization, and sentiment analysis.

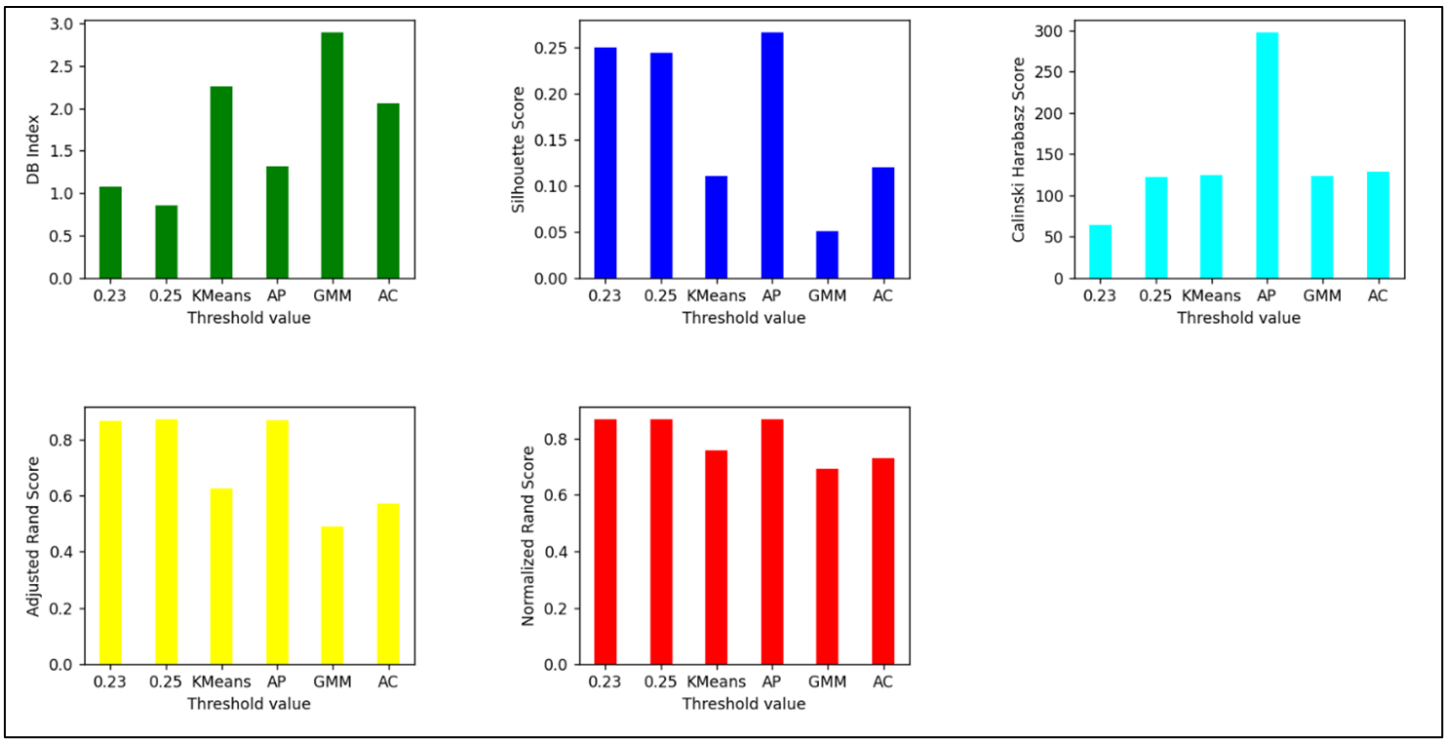

Figure 5: Caption for Graph 4

The Custom Clustering Algorithm, with threshold values of 0.23 and 0.25, demonstrated competitive performance compared to standard clustering algorithms such as KMeans, Affinity Propagation, Gaussian Mixture, and Agglomerative Clustering. The Davies-Bouldin index, Silhouette score, Calinski Harabasz score, Adjusted Rand Score, and Normalized Mutual Information Score were used as metrics for evaluation.

Model	Davies-Bouldin index	Silhouette Score	Calinski Harabasz score	Adjusted Rand Score	Normalized Mutual Info Score
Custom Clustering Algorithm (0.23)	1.0755	0.2502	63.9523	0.8651	0.8684
Custom Clustering Algorithm (0.25)	0.8524	0.2445	122.0241	0.8721	0.8694
KMeans	1.0535	0.4173	318.6811	0.6050	0.7224
Affinity Propagation	1.3178	0.2664	297.9176	0.8666	0.8675
Gaussian Mixture	1.4542	0.2545	261.868	0.8390	0.8438
Agglomerative Clustering	1.0346	0.4144	314.3739	0.6346	0.7512

Table 2: Optimum Number of Clusters

4.2 Conclusion

The algorithm's strengths lie in its adaptability to varying document relationships through dynamic correlation-based optimization. By leveraging semantic relationships, the algorithm creates cohesive and meaningful clusters. However, a potential limitation is the sensitivity to the choice of the correlation threshold, which warrants further exploration.The Custom Clustering Algorithm holds promise in various applications where document organization and extraction of semantic patterns are crucial. Its ability to dynamically adjust to document relationships makes it suitable for datasets with diverse content and varying semantic associations. Future research could explore mechanisms for automated selection of correlation thresholds based on the dataset's characteristics. Additionally, scalability and robustness assessments on larger and more diverse datasets could further validate the algorithm's effectiveness.

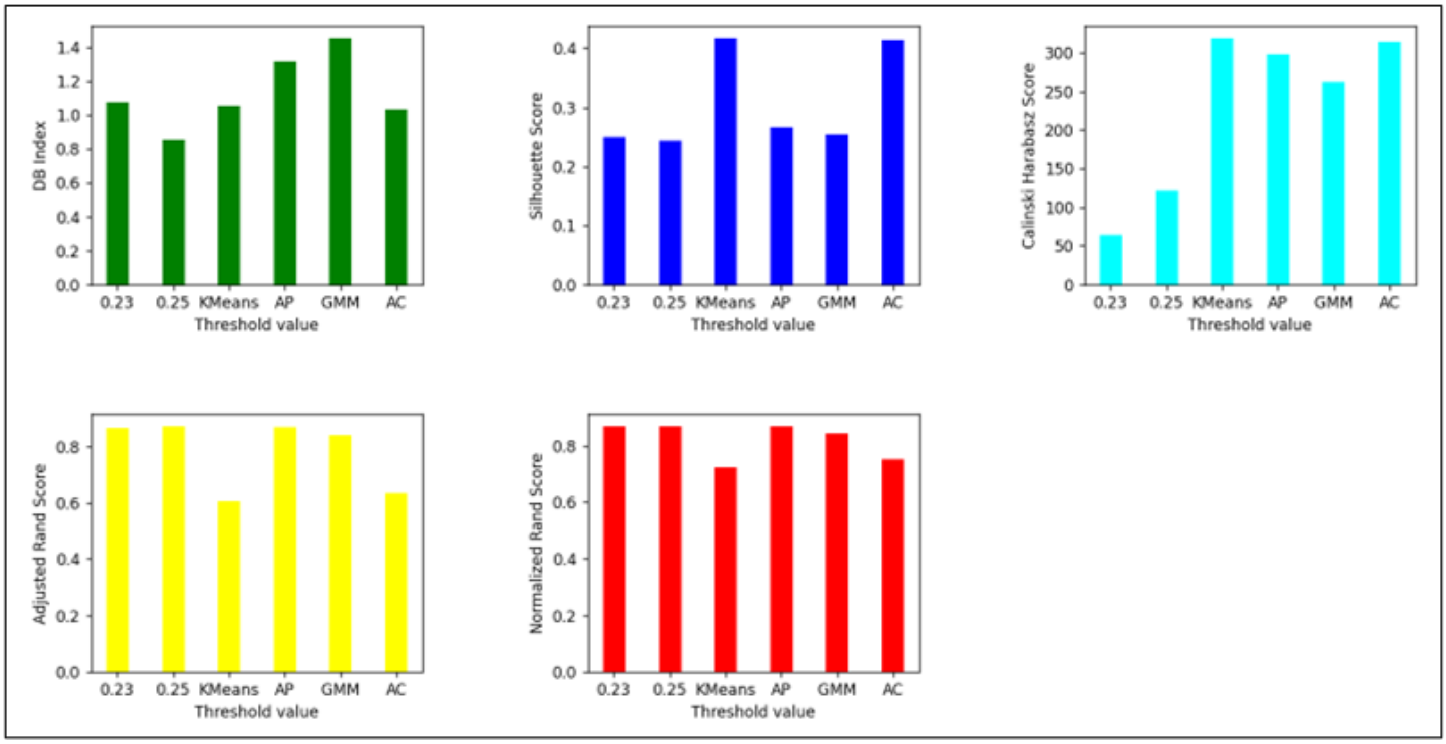

Figure 6: Caption for Graph 4

References

[1] Adel R Alharbi, Mohammad Hijji, and Amer Aljaedi. Enhancing topic clustering for arabic security news based on k-means and topic modelling. *IET Networks*, 10(6):278–294, 2021.

[2] Nino Arsov, Milan Dukovski, Blagoja Evkoski, and Stefan Cvetkovski. A measure of similarity in textual data using spearman's rank correlation coefficient. *arXiv preprint arXiv:1911.11750*, 2019.

[3] Austin J Brockmeier, Tingting Mu, Sophia Ananiadou, and John Y Goulermas. Self-tuned descriptive document clustering using a predictive network. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1929–1942, 2018.

[4] Jasmine Irani, Nitin Pise, and Madhura Phatak. Clustering techniques and the similarity measures used in clustering: A survey. *International journal of computer applications*, 134(7):9–14, 2016.

[5] R Jensi and Dr G Wiselin Jiji. A survey on optimization approaches to text document clustering. *arXiv preprint arXiv:1401.2229*, 2014.

[6] Mamta Kayest and Sanjay Kumar Jain. Optimization driven cluster based indexing and matching for the document retrieval. *Journal of King Saud University-Computer and Information Sciences*, 34(3):851–861, 2022.

[7] Claude Sammut and Geoffrey I. Webb, editors. *TF–IDF*, pages 986–987. Springer US, Boston, MA, 2010.

[8] Ferenc Tolner, Márta Takács, György Eigner, and Balázs Barta. Clustering of business organisations based on textual data - an lda topic modeling approach. In *2021 IEEE 21st International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 000073–000078, 2021.

[9] Dewan F Wahid and Elkafi Hassini. A literature review on correlation clustering: Cross-disciplinary taxonomy with bibliometric analysis. In *Operations Research Forum*, volume 3, page 47. Springer, 2022.