Extractive Text Summarization Using Word Similarity Based Spectral Clustering

Fahim Morshed, Md. Abdur Rahman, Sumon Ahmed

October 12, 2024

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1 Introduction

Text Summarization is the process of shortening a larger text without losing any key information. Automatic Text Summarization (ATS) is the process of summarizing a document automatically [29]. This is a major area of natural language processing (NLP) research which is progressing very rapidly. This task has become more and more important in the current digital age, where the amount of textual data grows exponentially in many fields [16], such as news, legal documents, health reports, research papers, and social media content. ATS techniques allow users to quickly get the essential information without needing to read through large amounts of text [7]. Automatic Text summarization is used in many fields, from automatic news summarization, content filtering, and recommendation systems to assisting legal professionals in going through long documents and researchers in reviewing academic papers. It can also play a critical role in personal assistants and chatbots, providing condensed information to users quickly and efficiently [27].

There are two main types of automatic text summarization: extractive and abstractive [27]. Extractive summarization, which is the focus of this paper, works by selecting best sentences or phrases directly from the source document, maintaining the original wording and sentence structure [20]. In contrast, abstractive summarization involves generating new sentences to capture the meaning of the text, Similar to human made summarization [19]. Extractive summarization is widely used because of its simplicity and effectiveness, especially for languages with limited NLP resources [11].

There are a lot of different ways to achieve extractive summarization. A commonly used method for extractive text summarization is graph-based summarization. This method represents the sentences of a document as nodes of a graph, and the edges between them are weighted by the similarity between the sentences [7]. Popular algorithms like LexRank [8] and TextRank [17] build graphs based on cosine similarity between sentence embeddings and apply ranking algorithms such as PageRank [21] in case of LexRank [8] or Random Walk in case of TextRank [17] to determine which sentences are the most important. These sentences are then selected to make the summary. Graph-based methods offer a more robust way to capture sentence importance and relationship, ensuring that the extracted summary covers the key information while minimizing redundancy [7].

A subset of graph-based approach to extractive summarization is clustering-based summarization. Here, sentences are grouped into clusters based on their semantic similarity, and one representative sentence from each cluster is chosen to form the summary [18]. Clustering reduces redundancy by ensuring that similar sentences are grouped together and that only the most representative sentence is selected. This method is effective in documents that cover multiple topics or subtopics, as it allows the summary to touch on each area without being repetitive.

For Bengali, a low-resource language, early attempts at text summarization relied on traditional methods like TF-IDF (Term Frequency-Inverse Document Frequency) scoring [4, 26]. These approaches, while simple, faced challenges in capturing the true meaning of sentences, as they treated words as isolated terms [27]. Graph-based methods introduced improvements by incorporating sentence similarity, but they were still limited by the quality of the embeddings used for the Bengali language. With the advent of word embedding models like FastText [10], which supports over 157 languages, including Bengali, it became possible to represent words in a Vector Space Model, thus enabling more accurate sentence similarity calculations.

However, existing models that use word embeddings, such as Roychowdhury et al.'s [23] Sentence Average Similarity-based Spectral Clustering (SASbSC) method, encountered issues when averaging word vectors to represent sentence meaning. This method failed in most cases because words in a sentence are often complementary rather than being similar, leading to inaccurate sentence representations when averaging their vectors. As a result, important word-to-word relationships between sentences were lost, reducing the effectiveness of the method.

In this paper, we propose a new approach to address these challenges. Our method improves upon previous attempts [23] by focusing on the individual similarity between words in sentences rather than averaging word vectors. Here the gaussian similarities between each word and the Most Similar Word from the other sentence for that word are used to get the similarity between the two sentence. This method captures the similarity of meaning between two sentences more accurately. By applying Gaussian similarity to the Most Similar Word Distance (D_{msw}) values, we build an affinity matrix that better reflects sentence closeness which can be proved by the effectiveness of the model (Table 1). We then apply spectral clustering on this matrix to group similar sentences together and use TF-IDF to select the most representative sentences from each cluster. This approach reduces redundancy and improves the quality of the summary by selecting sentences that are not only relevant but also diverse. This method works really well for Bengali on four diverse datasets consistently (Figure 3). It consistently outperforms other graph based methods like BenSumm [3], SASbSC [23], LexRank [8]. It also performs similarly well on other low resource languages we tried it on. These languages are Hindi, Marathi and Turkish (Table 3). These are the only other low resource languages where we found reliable evaluation datasets and tested our model on them. The search process was not exhaustive due to our language barrier.

The main contribution of this paper are: (I) Creating a new way to calculate similarity between two sentence. (II) Contributes a novel methodology for extractive text summarization for the Bengali language. by improving sentence similarity calculations and enhancing clustering techniques. (III) It addresses the limitations of previous models, such as misleading word vector averages [23], (IV) It offers a Generalizable solution for creating less redundant and information rich summaries across languages. (V) It provides a publicly available high quality dataset of 500 human generated summary.

The rest of the paper is organized as follows: The Literature review and Methodology are described in section 2 and 3 respectively. The section 4 illustrates the findings of this work. The section 5 discusses the findings of the paper in more depth.

2 Literature Review

Text Summarization has been an important necessity for textual data consumption for a long time. But manually summarizing is really time-consuming and counter-productive. So automating the Text Summarization process has been a research problem for a long time. Attempts at automatic text summarization started with indexing-based methods [1]. In this attempt Baxendale [1] attempted to summarize text by scoring sentences higher based on a certain word list. But this type of method failed to capture the topic and essence of the input text. To solve this, Text Summarization with statistical methods like TF-IDF became very popular. Edmundson [6] proposed a method which can focus on the central topic of a document. It uses two metrics, Term Frequency (how many times a term appears in the input) and Inverse Document Frequency (inverse of how many documents the term appears in a corpus) to calculate the importance of a term in a document. This method identifies the words that are common in the input text but not as common in the language and identifying them as the central topic. But it was too error-prone due to it thinking every word as a unique isolated term and not having any semantic relation with other words. Some words may be a central topic of a document but not identified as such because they got divided into too many synonyms.

Modern breakthroughs into the extractive text summarization began with the usage of Graph-based Extractive Text Summarization methods like LexRank [8] or TextRank [17]. LexRank [8] calculates the similarity between two sentences using cosine similarity and builds a graph containing similarity between every pair of sentences in the input. The most important sentences are then identified using the PageRank [21] algorithm on the graph. This algorithm ranked the sentences, who are most similar with other high ranked sentences, higher. TextRank [17] also uses a similar approach, but for every sentence, the method distributed its scores to its neighbours using a random walk. The process was done over and over until the scores converge. Although these models are very novel compared to their time, they still lacked fundamental understanding of the words involved in a sentence.

To solve this problem by better expressing the similarity between words, a mathematical abstraction called Word Vector Embedding was conceptualized by the seminal work of Salton et al. [25]. Word Vector Space is a mathematical abstraction of a vocabulary where the closer two words are meaning-wise, the closer they are in the vector space. Using word vector for summarization has only been started to be attempted recently [14].

But Text Summarization attempts in Bengali are a more recent development than in other high resource languages. So, a lot of sophisticated approaches from other languages haven't been attempted yet. Earlier Extractive methods have been focused on some derivative of TF-IDF based text summarization such as Chowdhury et al. [3], Dash et al. [4], Sarkar [26]. Sarkar [26] used simple TF-IDF score of each sentence to rank them and pick the best sentences. Dash et al. [4] used weighted TF-IDF along with some other features like sentence position to rank the sentences. Chowdhury et al. [3] however, used TF-IDF matrix of a document to build a graph

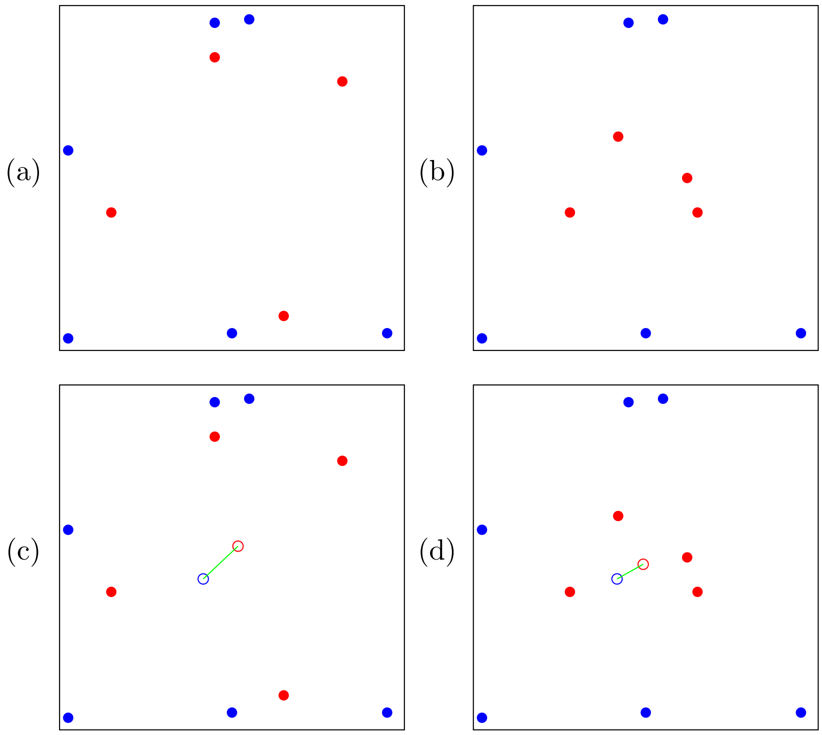


Figure 1: Scenarios where averaging method fails.

and perform Hierarchical Clustering to group sentences together and pick one sentence from each group. One shortcoming of this model is that TF-IDF matrix is not semantically equivalent to the actual sentences. So it didn't perfectly represent the sentences' semantic closeness in the graph. Using Word Vector Embedding for Bengali has solved this problem. FastText [10] released a dataset[1] that had word vector embedding in 157 languages, including Bengali. Using this dataset, Roychowdhury et al. [23] proposed a model where they replaced all the words with their respective vector, then averaged the vectors in a sentence to get the vector for a sentence. The Gaussian Similarity between the vectors is used to build the graph. On the graph, spectral clustering was used to group them together and pick one sentence from each cluster using cosine similarity to get the summary.

But this model also had a critical weakness. Words in a sentence do not have similar meaning, instead they express different parts of one whole meaning of a sentence. Which means they are complementary instead of being similar. So word averages always tend to be in the center and don't represent the semantic similarity anymore because the word vectors get scattered throughout the vector space due to this complementary nature. An example is shown in Figure 1 where the distance between the average word vectors is being misleading. In the figure, each point represents a word vector. The words from the same sentence are grouped together by being colored the same. In Figure 1(a), a scenario is shown in which the words of the two sentences are closer together in a vector space. The average distance between these two sentences can be seen in the Figure 1(c). We can see that averaging the words made both of the average clusters in the center. In Figure 1(b), we can see a different scenario where the word vectors are farther apart meaning wise. But the Figure 1(d) shows the average vector for these two sentences is closer than in the first scenario, thus making this metric misleading. This shortcoming has been one of the key motivations for this research.

[1] *https://fasttext.cc/docs/en/crawl-vectors.html*

3 Methodology

The summarization process followed here can be boiled down as, grouping together all the close sentences together based on their meaning and then picking one sentence from each group to minimize redundancy and maximize sentence coverage. Often most widely used extractive summarization methods involve scoring the sentences based on some metric and then picking the best scoring sentences to generate the summaries [4, 26]. But in a single topic text document it often picks similar sentences creating redundancy due to all of them having the central topic terms. To mitigate this, firstly grouped the sentences with similar meaning together and then finally picked one sentence from each group to generate a summary. This will ensure maximum coverage of topics while also reducing redundancy. This method has also been tried before by Roychowdhury et al. [23]. But the main challenge to reducing the redundancy is to develop a method that can accurately predict how close the meaning of the two sentences are. In this paper, we propose a method that can do it. The summarization process followed here involves total 4 steps. These are, in order of their use, Pre-processing, Sentence similarity calculation, Clustering and Summary generation. These steps are further discussed in the following subsections.

3.1 Pre-processing

Pre-processing is a standard step in Natural Language Processing that transforms the raw human language inputs into a format that can be used by a computer algorithm. Here the input document is transformed into a list of sets of vectors where each word is represented with a vector, each sentence as a set of vectors and the whole document as a list of said sets. The Pre-processing involves 3 steps. These are Tokenization, Stop Word Removal, Word Embedding. A very common step in Pre-processing, Word Stemming, isn't used in here as the word embedding dataset works best for the whole word instead of the stemmed word. These steps are further discussed bellow. This whole process is shown in Algorithm 1.

3.1.1 Tokenization

Tokenization is the step of dividing an input document into sentences and words in a usable format. Here the input document was firstly divided into sentences by using the NLTK library [2] of python by using regex matching. These sentences are then further divided into words and are put together in a list. Then these lists are further compiled into a list of list for the whole input document. An example of performing this step is given bellow.

Before Tokenization:
রাশিয়া-ইউক্রেন যুদ্ধ শুরু হওয়ার পর মার্কিন ডলারের বিনিময় হার বেড়েছে। এতে অমদানিনির্ভর দেশগুল বিপাকে পড়েছে। বৈদেশিক মুদ্রার রিজার্ভ বা মজুত কমে যাওয়ায় বাংলাদেশকেও অমদানি সীমিত করতে হয়েছে।

After Tokenization:
((রাশিয়া, ইউক্রেন, যুদ্ধ, শুরু, হওয়ার, পর, মার্কিন, ডলারের, বিনিময়, হার, বেড়েছে), (এতে, অমদানিনির্ভর, দেশগুল, বিপাকে, পড়েছে), (বৈদেশিক, মুদ্রার, রিজার্ভ, বা, মজুত, কমে, যাওয়ায়, বাংলাদেশকেও, অমদানি, সীমিত, করতে, হয়েছে))

3.1.2 Stop Word Removal

Stop words, such as prepositions and conjunctions, add sentence fluidity but don't carry significant meaning. For Bengali, a commonly used dataset[2] of 363 Bengali stop words used to remove the stop words on a matching basis. After completing this step the input document would look like this:

[2] *https://www.ranks.nl/stopwords/bengali*

((রাশিয়া, ইউক্রেন, যুদ্ধ, শুরু, মার্কিন, ডলারের, বিনিময়, হার, বেড়েছে), (অমদানিনির্ভর, দেশগুল, বিপাকে, পড়েছে), (বৈদেশিক, মুদ্রার, রিজার্ভ, মজুত, কমে, যাওয়ায়, বাংলাদেশকেও, অমদানি, সীমিত))
Here the removed stop words are হওয়ার, পর, এতে, বা, করতে, হয়েছে

3.1.3 Word Embedding

Word Embedding is a step that replaces the words in a sentence with a corresponding vector in a vector space such that the closer two words are in terms of meaning to one another the smaller the distance of those vectors would be. To achieve this step, we used a dataset with 1.47 million Bengali words produced by Grave et al. [10] crawling Wikipedia and other online resources to make the word embedding vectors. Finally, each word that is present in the tokenized and filtered list is replaced with their corresponding vectors and the words that isn't found is ignored and considered to be too rare to be relevant.

Algorithm 1 Preprocessing

SentenceList ← tokenize(D)
for each sentence i in SentenceList **do**
    WordList.append(tokenize(sentence$_i$))
end for
VectorList ← {}
for each WordList$_i$ in WordList **do**
    VectorList$_i$ ← {}
    for each Word in WordList$_i$ **do**
        if Word ∈ StopWordList **then**
            WordList$_i$.remove(Word)
        else if Word ∈ VectorEmbedding.keys() **then**
            VectorList$_i$.append(VectorEmbedding.get(Word))
        else
            WordList$_i$.remove(Word)
        end if
    end for
    VectorList.append(VectorList$_i$)
end for
Return VectorList

3.2 Sentence Similarity Calculation

To perform clustering in a graph, an affinity matrix is needed. A similarity calculation technique using individual word distance and Gaussian similarity have been proposed here. The process is shown in Algorithm 2. A previous similar method by Roychowdhury et al. [23] had averaged all the vectors present in a sentence to get a vector for the sentence. But this is not rally a sound strategy as words in a sentence are generally complementary instead of being similar. So the word vectors tend to scatter around in the vector space rather than being grouped closed together. This leads to the average word vector having a tendency towards the center of the vector space. This would lead to the affinity between these vectors not being representative of the actual meaning.

To mitigate this, in this study, similarity between individual words in a pair of sentences have been considered. For this, firstly, the Most Similar Word Distance (D_{msw}) have to be calculated as shown in Equation 1. The D_{msw} denotes the distance between a word vector and the word vector that is closest to its meaning from the other sentence.

$$D_{msw}(x, Y) = min(\{d(x, y_i) : y_i \in Y\}) \tag{1}$$

All the D_{msw} for each word in each sentence are then put together in a list like shown in Equation 2.

$$D_{msw} = \{D_{msw}(x, Y) : x \in X\} \cup \{D_{msw}(y, X) : y \in Y\} \tag{2}$$

Here, for every word vector x in a sentence X, the closest vector, in terms of Euclidean distance denoted by $d(x, y)$, is identified from all the word vectors y in the sentence Y. Secondly, the word similarity is calculated using Gaussian similarity for each of these D_{msw}. The equation involved is shown in Equation 3.

$$WSim_i = e^{\frac{-D_{msw_i}^2}{2\sigma^2}} \tag{3}$$

The Sentence similarity between the two sentence or $Sim(X, Y)$ is calculated as the Geometric mean of all the word similarities from both sentence so that the similarity between two sentence is symmetric. This is explained in the Equation 4

$$\begin{aligned} Sim(X, Y) &= \left(\prod_{i=1}^{n} WSim_i\right)^{\frac{1}{n}} \\ &= \left(e^{\frac{-D_{msw_1}^2}{2\sigma^2}} \cdot e^{\frac{-D_{msw_2}^2}{2\sigma^2}} \cdot \ldots \cdot e^{\frac{-D_{msw_n}^2}{2\sigma^2}}\right)^{\frac{1}{n}} \\ &= exp\left(-\frac{D_{msw_1}^2 + D_{msw_2}^2 + \ldots + D_{msw_n}^2}{2n\sigma^2}\right) \\ &= exp\left(-\frac{\sum_{i=1}^{n} D_{msw_i}^2}{2n\sigma^2}\right) \end{aligned} \tag{4}$$

Here, by taking geometric mean of the similarity between the closest words together in two sentence, an effective word to word comparison has been created between those sentences. This reduces any misleading distance that would have come from the word averaging method due to the tendency towards center. An example of this can be seen at Figure 2. Here, a more representative word association can be seen for both scenarios from Figure 1. Red and Blue dots in the figure represent two set of word vectors in a sentence pair. Black dashed lines show the Most Similar Distance ($D_{msw}(x, Y)$) for a word vector x and the other sentence Y. The arrowheads point from x. The Figure 2(a) shows the D_{msw} for Scenario A in Figure 1(a). The Figure 2(b) Shows the D_{msw} for Scenario B in Figure 1(b). We can see that problem caused by the averaging method have been mitigated here.

The similarity equation (equation-4) has a standard deviation σ which works as a control variable and was fine-tuned to be 5×10^{-11} where it gave the best results.

3.3 Clustering

The clustering is the most integral part of this summarization technique, aiming to group all the sentences with similar meanings together. Here, spectral clustering is used to cluster the sentences using sentence similarity calculated in the step above. Spectral clustering was chosen here because Roychowdhury et al. [23] found it to be better performing than DBSCAN method. The spectral clustering steps were followed according to the tutorial given by [28].

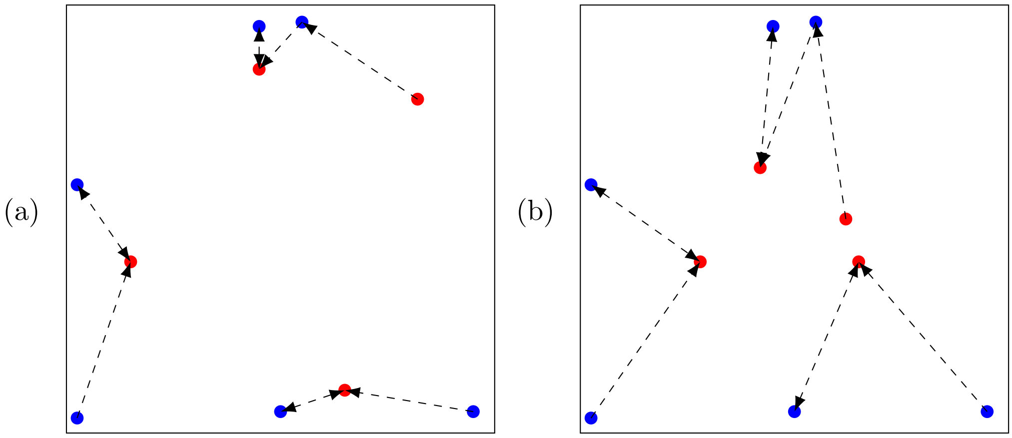


Figure 2: Process of obtaining D_{msw}

Algorithm 2 Sentence Similarity Calculation

$n \leftarrow \text{length}(VL)$
$A \leftarrow \{\{0\} \times n\} \times n$
for each sentence$_i$ in VL **do**
  $D_{\text{Square}} \leftarrow 0$
  count $\leftarrow 0$
  for each sentence$_j$ in VL **do**
    for each word$_i$ in sentence$_i$ **do**
      $D_{\text{msw}} \leftarrow \infty$
      for each word$_j$ in sentence$_j$ **do**
        if Distance(word$_i$, word$_j$) $< D_{\text{msw}}$ **then**
          $D_{\text{msw}} \leftarrow$ Distance(word$_i$, word$_j$)
        end if
      end for
      $D_{\text{Square}} \leftarrow D_{\text{Square}} + D^2_{\text{msw}}$
      count++
    end for
    for each word$_j$ in sentence$_j$ **do**
      $D_{\text{msw}} \leftarrow \infty$
      for each word$_i$ in sentence$_i$ **do**
        if Distance(word$_i$, word$_j$) $< D_{\text{msw}}$ **then**
          $D_{\text{msw}} \leftarrow$ Distance(word$_i$, word$_j$)
        end if
      end for
      $D_{\text{Square}} \leftarrow D_{\text{Square}} + D^2_{\text{msw}}$
      count++
    end for
    similarity $\leftarrow \exp\left(\frac{-D_{\text{Square}}}{2 \times \text{count} \times \sigma^2}\right)$
    $A[i][j] \leftarrow A[j][i] \leftarrow$ similarity
  end for
end for
Return A

To perform spectral clustering on a data, firstly, an affinity matrix is required that shows the weight of edges between the vertexes in the graph. Here the affinity A is prepared using the following Equation 5.

$$A_{ij} = A_{ji} = Sim(S_i, S_j) \tag{5}$$

Here, S_i, S_j are sentences from the input document. The affinity matrix, A, is used in the spectral clustering implementation in the SciKit-learn tool [22] in python. Here, to use the function, we also need to provide the number of clusters to achieve. The number of clusters are fixed at $k = ceiling\left(\frac{N}{5}\right)$ due to it being a reasonable size to contain all necessary sentences as well as being short enough to be an effective summary.

3.4 Summary Generation

After clustering, we pick one sentence from each cluster. The sentences inside a cluster are ranked among themselves using TF-IDF . The best ranked sentence of the clusters by their order of TF-IDF score is selected from each cluster. We then rearranged these picked sentences are in their order of appearance to retain the normal flow of information in the input. These sentences are then concatenated together to produce the final output summary. This is shown in Algorithm 3

Algorithm 3 Summary Generation

$k \leftarrow \lceil$ length(A) / 5 $\rceil$
clusters $\leftarrow$ spectral_clustering(adjacency = A, k)
indexes $\leftarrow$ {}
for each cluster$_i$ in clusters **do**
    TFIDF $\leftarrow$ {}
    for each index in cluster$_i$ **do**
        TFIDF.append(tfidf_sum(sentences[index]))
    end for
    indexes.append(indexof(max(TFIDF)))
end for
sort(indexes)
$S \leftarrow$ ""
for each i in indexes **do**
    $S \leftarrow S+$ sentences[i]
end for
Return S

4 Result

The text summarization performance of the proposed model is compared against the BenSumm [3], LexRank [8] and Sentence Average Similarity-based Spectral Clustering(SASbSC)-based summarization method [23] methods. These methods are the recently published state of the art model for Bengali Extractive Text Summarization. A classic extractive text summarizing method LexRank [8] was also used as a benchmark for comparison.

4.1 Evaluation Datasets

To examine our proposed model, we compared our model along with the 3 benchmark models on 4 different diverse datasets. We did this so that the results doesn't become biased due to any problem with the dataset.

4.1.1 Dataset-1 (Self-curated)

To evaluate the performance of implemented text summarization methods [3, 8, 23], a curated Bengali extractive text summarization dataset was produced by an expert linguistic team. 250 news documents of various sizes were summarized for this purpose. Each document was summarized twice by two different person to minimize human bias. In total, there is 500 different document-summary pair in this dataset. This dataset is made publicly available[3] for other researchers to use for evaluation purpose in their research.

4.1.2 Dataset-2 (Towhid Ahmed Foysal)

This dataset is a collection of summary article pair from The Daily Prothom Alo. It was published by Towhid Ahmed Foysal in Kaggle[4]. The original dataset was filtered so that all the articles smaller than 50 characters and all the summaries that contains something not in the original articles were discarded. After filtering, total 10,204 articles remained, each with 2 summaries.

4.1.3 Dataset-3 (BNLPC)

This dataset is a collection of news article summaries published by Haque et al. [12]. The dataset was collected from GitHub[5]. The dataset contains 100 article with 3 different summaries for each article.

4.1.4 Dataset-4 (Abid Mahdi)

This dataset was published by Abid Mahdi on GitHub[6]. The dataset contains 200 documents each with 2 summaries.

4.2 Text Summarization Models

Four different Bengali extractive text summarization models were implemented to evaluate them by comparing the machine generated summaries against the human generated summaries from the datasets described above.

Model-1: Model-1 is the proposed model for this paper. The model uses word vector based Gaussian similarity to perform spectral clustering to group similar sentences together and extract one sentence from each group. This is described as Word Similarity based Spectral Clustering (WSbSC)

Model-2: Model-2 (SASbSC) is the method proposed by Roychowdhury et al. [23]. This extractive text summarization method is similar to the proposed method. SCSbSC method uses the same word embedding dataset as ours to get word vectors. It uses a sentence center similarity based graph to perform spectral clustering inside the method. Then use cosine similarity to extract sentences from the input. To get sentence similarity, SCSbSC averages all the word vectors of a particular sentence to get the Sentence center. This method was implemented in python as described in their article.

[3] *dataset link*
[4] *https://www.kaggle.com/datasets/towhidahmedfoysal/bangla-summarization-datasetprothom-alo*
[5] *https://github.com/tafseer-nayeem/BengaliSummarization/tree/main/Dataset/BNLPC/Dataset2*
[6] *https://github.com/Abid-Mahadi/Bangla-Text-summarization-Dataset*

Model-3: BenSumm describes two different summarization method in the study [3]. Here only the extractive method is implemented and compared because the proposed method is also extractive in nature. BenSumm implements a TF-IDF based cosine similarity graph between the sentences and then clusters the sentences using Agglomerative Clustering. The implementation codes are publicly available in GitHub[7].

Model-4: LexRank [8] uses a TF-IDF based Matrix and Googles PageRank algorithm [21] to rank sentences. The top ranked sentences are selected and arranged into summary after that. An implemented version of this method is available as a python package in PyPI as LexRank[8]. LexRank is applied using a large Bengali Wikipedia corpus[9].

4.3 Evaluation Metrics

To evaluate the correctness of the machine generated summaries compared to the human generated summaries, we used the ROUGE method [15]. It compares a human produced reference summary with a machine generated summary. The ROUGE method uses N-gram based overlapping to find a recall, precision and F-1 score. The ROUGE implementation that were used is available as a python package in PyPI[10]. There are three different metrics in the package for comparison of the summaries. These are:

1. **ROUGE-1:** It uses unigram matching to find how much similar two summaries are. It is a good first impression for performance but can be misleading too as many large enough texts will share very high proportion of uni-grams between them.

2. **ROUGE-2:** It uses bi-gram matching to find how much similar the two summaries are in a word level. Shared bigrams lead to a deeper analysis of syntactic similarities between the two summaries.

3. **ROUGE-LCS:** It finds the longest common sub-sequence between the summaries to calculate the rouge scores. It can calculate the similarity in flow of the sentences between two summaries.

In this study, we compared the F-1 scores from each of these metrics for the 4 models.

4.4 Comparison

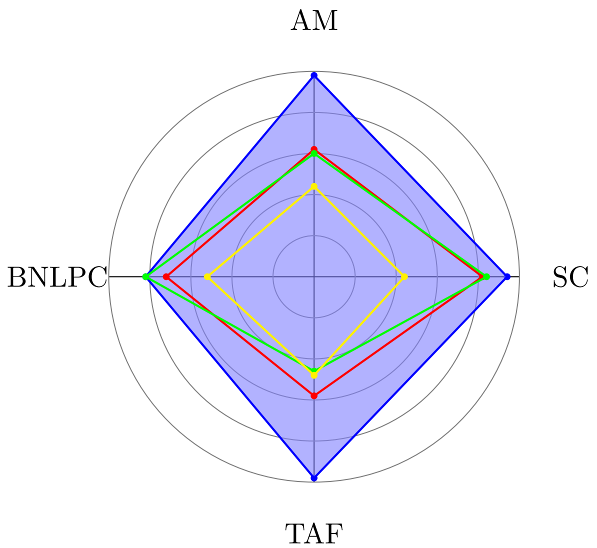
Average F-1 scores for the three Rouge metrics (Rouge-1, Rouge-2, Rouge-LCS) of the four models(Proposed, SASbSC, BenSumm, LexRank) on the 4 datasets are shown in the table-1.

These results are further summarized into 3 radar charts so that the performance of each model on each metric for the datasets can be visualized.
These charts (Figure 3) shows us that the proposed method is much more dataset independent and performs uniformly on every metric across the datasets. Other models although performs good on certain datasets, fails to show consistency.

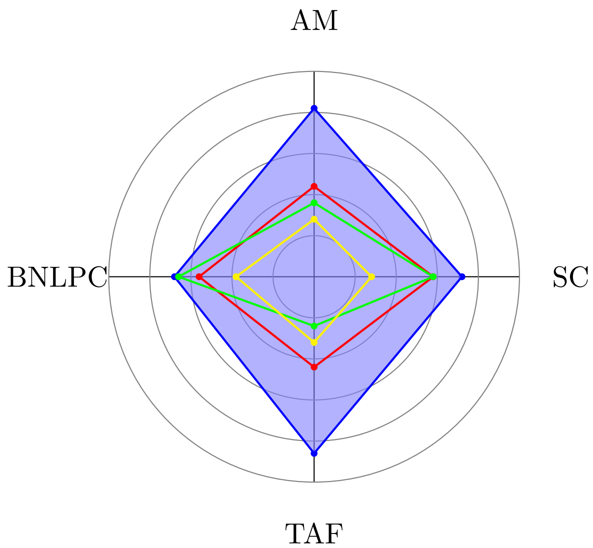
4.5 Different Ranking Techniques Inside Clusters

We implemented two ranking methods to pick the best sentence from each clusters. First one is the First Rank method where we just pick the sentence that is first in terms of their order

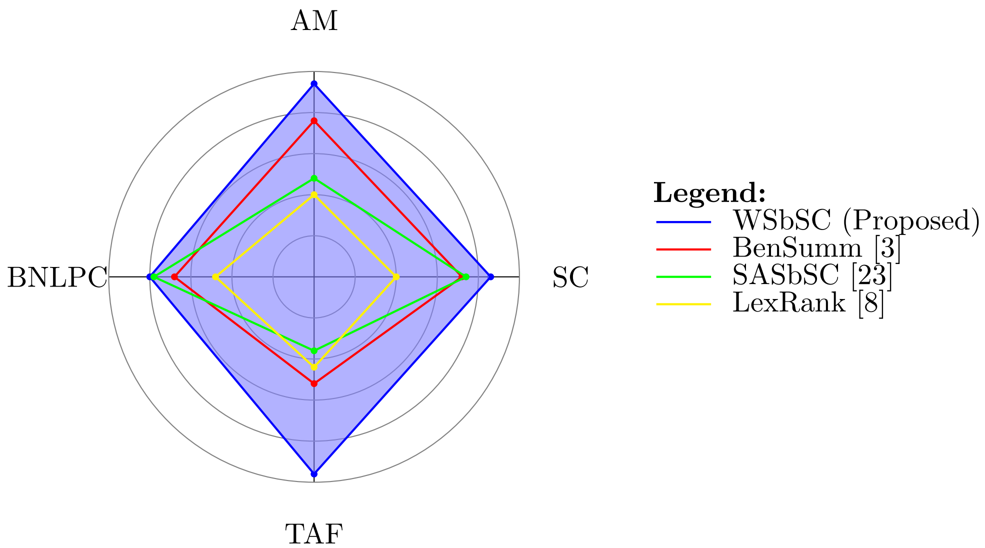
[7] *https://github.com/tafseer-nayeem/BengaliSummarization*
[8] *https://pypi.org/project/lexrank/*
[9] *https://www.kaggle.com/datasets/shazol/bangla-wikipedia-corpus*
[10] *https://pypi.org/project/rouge/*



(a) Comparison of ROUGE-1 Values



(b) Comparison of ROUGE-2 Values



(c) Comparison of ROUGE-LCS Values

Figure 3: Radar chart of the models being compared on 3 different metrics and 4 datasets.

Dataset-1 (SC)			
Model	Rouge-1	Rouge-2	Rouge-LCS
Model-1 (WSbSC)(Proposed)	**0.47**	**0.36**	**0.43**
Model-2 (BenSumm) [3]	0.41	0.29	0.36
Model-3 (SASbSC) [23]	0.42	0.29	0.37
Model-4 (LexRank) [8]	0.22	0.14	0.20
Dataset-2 (TAF)			
Model-1 (WSbSC)(Proposed)	**0.49**	**0.43**	**0.48**
Model-2 (BenSumm) [3]	0.29	0.22	0.26
Model-3 (SASbSC) [23]	0.23	0.12	0.18
Model-4 (LexRank) [8]	0.24	0.16	0.22
Dataset-3 (BNLPC)			
Model-1 (WSbSC)(Proposed)	**0.41**	**0.34**	**0.40**
Model-2 (BenSumm) [3]	0.36	0.28	0.34
Model-3 (SASbSC) [23]	**0.41**	0.33	0.39
Model-4 (LexRank) [8]	0.26	0.19	0.24
Dataset-4 (AM)			
Model-1 (WSbSC)(Proposed)	**0.49**	**0.41**	**0.47**
Model-2 (BenSumm) [3]	0.31	0.22	0.28
Model-3 (SASbSC) [23]	0.30	0.18	0.24
Model-4 (LexRank) [8]	0.22	0.14	0.20

Table 1: Comparison of average Rouge scores between graph based extractive summarization models on 4 different datasets

Method	Rouge-1	Rouge-2	Rouge-LCS
FirstRank	0.47	0.36	0.43
TF-IDF	**0.50**	**0.40**	**0.46**

Table 2: Comparison of Result of different ranking techniques

of appearance inside the input document. The second one is the TF-IDF ranking, where we ranked the sentences by their TF-IDF scores and pick the best one. We can see in the table 4.5 that the TF-IDF scores better on a high quality dataset like our Self-curated one.

4.6 Implementation Into Other Languages

The model discussed here is not language dependent. So the model can be easily extended into other languages. To perform this method into a language, we only need a language specific tokenizer, a list of stop-words and a word vector embedding dataset. We tried to find quality extractive text summarization dataset for evaluating the method, but could only find relevant datasets on 3 other languages. These are Hindi, Marathi and Turkish. We adopted this Model into these 3 low resource languages to check this hypothesis.

The Table-3 shows the result of the proposed word similarity based spectral clustering method for extractive summarization in other low resource languages. For the Hindi language, we used a Kaggle dataset[11] produced by Gaurav Arora. For the Marathi language we used another Kaggle dataset[12] produced by Ketki Nirantar. For the Turkish language we used a Github dataset[13]

[11] *https://www.kaggle.com/datasets/disisbig/hindi-text-short-and-large-summarization-corpus/*
[12] *https://www.kaggle.com/datasets/ketki19/marathi*
[13] *https://github.com/xtinge/turkish-extractive-summarization-dataset/blob/main/dataset/XTINGE-SUM_TR_EXT/xtinge-sum_tr_ext.json*

Language	Rouge-1	Rouge-2	Rouge-LCS
Bengali (Dataset - 1)	0.47	0.36	0.43
Bengali (Dataset - 2)	0.49	0.43	0.48
Bengali (Dataset - 3)	0.41	0.34	0.40
Bengali (Dataset - 4)	0.49	0.41	0.47
Bengali (Average)	0.47	0.38	0.44
Hindi	0.40	0.26	0.36
Marathi	0.50	0.42	0.50
Turkish	0.48	0.39	0.47

Table 3: Comparison of Result of proposed summarization method in other low-resource languages

produced by the XTINGE [5] team. We can see that the results remain very close despite the change in language.

5 Discussion

The results presented in the previous sections highlight the effectiveness of the proposed Word Similarity-based Spectral Clustering (WSbSC) model for extractive text summarization in Bengali, as well as its adaptability to other low-resource languages. This section delves into an analysis of the comparative results, the strengths and limitations of the proposed method, and potential areas for further research.

As evidenced by the results, the WSbSC model consistently outperforms the baseline models, namely BenSumm [4], LexRank [8], and Sentence Average Similarity-based Spectral Clustering (SASbSC) [23], across multiple datasets. This performance improvement is largely for the novel approach of calculating sentence similarity based on the geometric mean of individual word similarities, which overcomes the problems of averaging methods that tend to pull sentence vectors into the center of the vector space. The Gaussian similarity-based approach used in WSbSC provides a more novel and precise method for capturing the semantic relationships between sentences.

The superior performance of WSbSC is especially noticeable in terms of ROUGE scores, where it consistently achieves higher F-1 scores across all datasets (Table 1). This suggests that the proposed method generates summaries that are more faithful to the human written reference summaries. Additionally, the clustering approach makes sure that sentences with similar information are grouped together, allowing the model to pick the most representative sentence from each group. This reduces redundancy and increases topic coverage, key components of a good summary.

The sentence similarity method proposed in this paper is a novel algorithm. Our proposed strategy is more suited for the job than other strategies can compare two sets of vectors such as Earth Movers Distance (EMD), Hausdorff Distance, Procrustes Analysis etc.. EMD [24] tries to find the lowest amount of "work" needed to transform one set into the other one. It considers adding a new point, removing a point, scaling the whole set, moving a point, rotating the set etc. as "work". This is very computationally expensive. And it also focuses on scaling and rotating which are not relevant in word vector space. Hausdorff distance [13] takes the worst case scenario and calculates the farthest distance between two points in the two set. It is easily influenced by outliers. It was avoided because words tend to spread out over the whole word space and this would suffer from the same problem as the averaging method. Procrustes

Analysis [9] tries to find the lowest amount of misalignment after scaling and rotating the two sets. Both of these processes are irrelevant in the context of word vector.

On the other hand, the proposed method focuses on Local Point Correspondence between two sets which is more important for words. The Gaussian similarity function captures the proximity of points smoothly, providing continuous feedback on how similar two points are in a normalized way. It is also robust against small outliers because of the use of a soft similarity measure (Gaussian) and geometric mean which helps smooth over small differences in point locations.

One of the key strength of this proposed method is the reduction of redundancy which is a common issue in extractive summarization methods. By grouping sentences with similar meanings and selecting a representative sentence from each group, the model ensures that the summary covers a broad range of topics without repeating itself. Our proposed model also has an improved sentence similarity calculation technique. Using the geometric mean of individual word similarities offers a more precise measure of how closely two sentences are related. This is a marked improvement over traditional methods that rely on word averaging, which often dilute the semantic meaning of a sentence. Another key strength is that it is found to be scalable across languages. By requiring only a language-specific tokenizer, stop-word list, and word embedding dataset, WSbSC can be easily adapted to other languages, as demonstrated in the experiments with Hindi, Marathi, and Turkish datasets (Table 3). This makes the model highly versatile and valuable for extractive summarization in low-resource languages.

Despite its advantages, the WSbSC model does face some challenges. The model heavily relies on pre-trained word embeddings, which may not always capture the full details of certain domains or newly coined terms. The FastText [10] dataset used here is heavily reliant on wikipedia for training. Which could introduce some small inherent unforeseen biases. In cases where the word embeddings do not fully have some word of a given document, the model's performance could degrade as it leaves those words out. The model also does not take into account the order in which words appear in a sentence or when the form special noun or verb groups. So it can be a little naive in some highly specialized fields. The calculation of word-to-word similarities and the subsequent clustering process can be computationally expensive, particularly for longer documents or datasets with a large number of sentences. While this issue is mitigated by the scalability of modern computing resources, it could still pose a limitation in resource-constrained environments. The decision to set the number of clusters to $\lceil N/5 \rceil$ (where N is the number of sentences) might not always yield optimal results for every document. In some cases, documents may contain many subtopics, needing more clusters to fully represent the diversity of content. Similarly, documents with a narrow focus may benefit from fewer clusters.

The WSbSC model has demonstrated its ability to perform well in low-resource languages such as Hindi, Marathi, and Turkish. Despite differences in language structure, the model's core methodology remained effective, yielding results that were consistent with the Bengali dataset evaluations. This underscores the potential of WSbSC as a generalizable approach for extractive summarization across different languages, provided that appropriate pre-processing tools and word embedding datasets are available.

The proposed Word Similarity-based Spectral Clustering model represents a significant advancement in Bengali extractive text summarization. Its ability to accurately capture sentence similarity, reduce redundancy, and generalize across languages makes it a valuable tool for summarizing text in low-resource languages. While there are still challenges to be addressed, the results of this study demonstrate the robustness and adaptability of the WSbSC model, offering a promising direction for future research in multilingual extractive summarization.

References

[1] P. B. Baxendale. Machine-made index for technical literature—an experiment. *IBM Journal of Research and Development*, 2(4):354–361, 1958. doi: 10.1147/rd.24.0354.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* "O'Reilly Media, Inc.", 2009.

[3] Radia Rayan Chowdhury, Mir Tafseer Nayeem, Tahsin Tasnim Mim, Md. Saifur Rahman Chowdhury, and Taufiqul Jannat. Unsupervised abstractive summarization of Bengali text documents. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2612–2619. Association for Computational Linguistics, April 2021. doi: 10.18653/v1/2021.eacl-main.224.

[4] Satya Ranjan Dash, Pubali Guha, Debasish Kumar Mallick, and Shantipriya Parida. Summarizing bengali text: An extractive approach. In *Intelligent Data Engineering and Analytics*, pages 133–140. Springer Nature Singapore, 2022. ISBN 978-981-16-6624-7.

[5] İrem Demir, Emel Küpçü, and Alptekin Küpçü. Extractive summarization data sets generated with measurable analyses. In *Proceedings of the 32nd IEEE Conference on Signal Processing and Communications Applications*, 2024.

[6] H. P. Edmundson. New methods in automatic extracting. *J. ACM*, 16(2):264–285, apr 1969. ISSN 0004-5411. doi: 10.1145/321510.321519.

[7] Wafaa S. El-Kassas, Cherif R. Salama, Ahmed A. Rafea, and Hoda K. Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2020.113679.

[8] Günes Erkan and Dragomir R. Radev. Lexrank: graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, dec 2004. ISSN 1076-9757.

[9] J. C. Gower. Generalized procrustes analysis. *Psychometrika*, 40(1):33–51, Mar 1975. ISSN 1860-0980. doi: 10.1007/BF02291478. URL `https://doi.org/10.1007/BF02291478`.

[10] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA), May 2018. URL `https://aclanthology.org/L18-1550`.

[11] Vishal Gupta and Gurpreet Lehal. A survey of text summarization extractive techniques. *Journal of Emerging Technologies in Web Intelligence*, 2, 08 2010. doi: 10.4304/jetwi.2.3.258-268.

[12] Md. Majharul Haque, Suraiya Pervin, and Zerina Begum. Automatic bengali news documents summarization by introducing sentence frequency and clustering. In *2015 18th International Conference on Computer and Information Technology (ICCIT)*, pages 156–160, 2015. doi: 10.1109/ICCITechn.2015.7488060.

[13] Felix Hausdorff. *Grundzüge der mengenlehre*, volume 7. von Veit, 1914.

[14] Aditya Jain, Divij Bhatia, and Manish K Thakur. Extractive text summarization using word vector embedding. In *2017 International Conference on Machine Learning and Data Science (MLDS)*, pages 51–55, 2017. doi: 10.1109/MLDS.2017.12.

[15] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81. Association for Computational Linguistics, July 2004. URL https://aclanthology.org/W04-1013.

[16] Bernard Marr. Big data: 20 mind-boggling facts everyone must read. https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-ever 2015. Accessed: 2024-10-12.

[17] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411. Association for Computational Linguistics, July 2004. URL https://aclanthology.org/W04-3252.

[18] G. Bharathi Mohan and R. Prasanna Kumar. A comprehensive survey on topic modeling in text summarization. In *Micro-Electronics and Telecommunication Engineering*, pages 231–240. Springer Nature Singapore, 2022. ISBN 978-981-16-8721-1.

[19] N. Moratanch and S. Chitrakala. A survey on abstractive text summarization. In *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pages 1–7, 2016. doi: 10.1109/ICCPCT.2016.7530193.

[20] N. Moratanch and S. Chitrakala. A survey on extractive text summarization. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–6, 2017. doi: 10.1109/ICCCSP.2017.7944061.

[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking : Bringing order to the web. In *The Web Conference*, 1999. URL https://api.semanticscholar.org/CorpusID:1508503.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] Sohini Roychowdhury, Kamal Sarkar, and Arka Maji. Unsupervised Bengali text summarization using sentence embedding and spectral clustering. In *Proceedings of the 19th International Conference on Natural Language Processing (ICON)*, pages 337–346. Association for Computational Linguistics, December 2022. URL https://aclanthology.org/2022.icon-main.40.

[24] Y. Rubner, C. Tomasi, and L.J. Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 59–66, 1998. doi: 10.1109/ICCV.1998.710701.

[25] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, nov 1975. ISSN 0001-0782. doi: 10.1145/361219.361220. URL https://doi.org/10.1145/361219.361220.

[26] Kamal Sarkar. An approach to summarizing bengali news documents. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, page 857–862. Association for Computing Machinery, 2012. ISBN 9781450311960. doi: 10.1145/2345396.2345535.

[27] Oguzhan Tas and Farzad Kiyani. A survey automatic text summarization. *PressAcademia Procedia*, 5(1):205–213, 2017. doi: 10.17261/Pressacademia.2017.591.

[28] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec 2007. ISSN 1573-1375. doi: 10.1007/s11222-007-9033-z. URL `https://doi.org/10.1007/s11222-007-9033-z`.

[29] Adhika Pramita Widyassari, Supriadi Rustad, Guruh Fajar Shidik, Edi Noersasongko, Abdul Syukur, Affandy Affandy, and De Rosal Ignatius Moses Setiadi. Review of automatic text summarization techniques & methods. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1029–1046, 2022. ISSN 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2020.05.006.