TEXT SUMMARIZATION USING NLP

A Project Report
Submitted for the Degree of

Bachelor of Technology in (CSE) of

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

by

JOEL T AJU	**ATP20CS030**
THEJAS KRISHNA P	**ATP20CS049**
VARUN V	**ATP20CS052**

Under the guidance of

DR. S GUNASEKARAN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHALIA SCHOOL OF ENGINEERING & TECHNOLOGY

Ahalia Campus , Palakkad , Kerala-67855

(http://www.ahalia.ac.in/)

June 27,2023

AHALIA SCHOOL OF ENGINEERING & TECHNOLOGY
ISO 9001:2015 Certified Institution. Approved by AICTE & Affiliated to A. P. J. Abdul Kalam Technological University
Ahalia Health, Heritage & Knowledge Village, Palakkad - 678557. Ph: 04923-226666, www.ahalia.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the report entitled "**TEXT SUMMARIZATION USING NLP**" submitted by **JOEL T AJU (ATP20CS030), THEJAS KRISHNA P (ATP20CS049), VARUN V (ATP20CS052)** for the award of the **Degree of Bachelor of Technology** in the **APJ Abdul Kalam Technological University** is a bonafide record of the work carried out by them under my guidance and supervision at Ahalia School of Engineering & Technology.

Guide:

Dr. S Gunasekaran

Head of the Department :

Dr. S Gunasekaran

Internal Examiner

External Examiner

AHALIA SCHOOL OF ENGINEERING & TECHNOLOGY
ISO 9001:2015 Certified Institution. Approved by AICTE & Affiliated to A. P. J. Abdul Kalam Technological University
Ahalia Health, Heritage & Knowledge Village, Palakkad - 678557. Ph: 04923-226666, www.ahalia.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **JOEL T AJU (ATP20CS030), THEJAS KRISHNA P (ATP20CS049), VARUN V (ATP20CS052)** hereby declare that this Project entitled "**TEXT SUMMARIZATION USING NLP**" is the record of the original work done by us under the guidance of "**Dr. S Gunasekaran",PhD**, Professor and

Dr. S.GUNASEKARAN The Head of Department, Professor, Department of Computer Science and Engineering, Ahalia School of Engineering and Technology. To the best of my knowledge, this work has not formed the basis for the award of any degree/diploma/ associateship/fellowship/or a similar award to any candidate in any University.

Place: **Signature of the Student**

Date:

COUNTERSIGNED

Dr. S.GUNASEKARAN

Professor & HoD

Department of CSE

ACKNOWLEDGEMENT

First and foremost we would like to thank the **GOD ALMIGHTY** for this infinite grace and help, without which this project would not have reached its completion. We wish to express our sincere gratitude to **Dr. P R Sreemahadevan Pillai M, Ph.D.**, Principal, Ahalia School of Engineering and Technology for his timely support throughout the course of this project. We extend our sincere thanks **to Dr. Krishna Kumar Kishor Ph.D**., Vice Principal, for extending all facilities.

We express our sincere and heartfelt gratitude to **Dr. S.Gunasekaran, Ph.D.**, Professor and Head of the Department, Department of Computer Science and Engineering for his help and excellent encouragement throughout the project. We express our deepest heartiest and sincere thanks to our Project Guide **Dr. S.Gunasekaran, Ph.D** ,Professor, for his exhilarating supervision, timely suggestions, and encouragement during all phases of this work. We take this opportunity to thank our Project Coordinator **Dr. S.Gunasekaran, Ph.D** Professor and our mentor who had been always there with us. Also, we would like to thank all the Teaching faculty members and Supporting Staff of our department.

We would like to convey our gratitude to our **Parents** whose prayers and blessings were always with us. Last but not least we would like to thank **our friends and others** who directly or indirectly helped us in the successful completion of this work.

TABLE OF CONTENT

ABSTRACT

Text summarization is an automated process that aims to generate a concise summary comprising important sentences, encompassing all relevant information from the original document. Two primary approaches in summarization, namely Extractive and Abstractive, have been observed based on the summary outcomes. While Extractive summarization has reached a certain level of maturity, current research endeavors have shifted focus towards Abstractive summarization and real-time summarization.

Over the past decade, there have been significant advancements in the acquisition of datasets, methods, and techniques in the field. The exponential growth of data on the Internet necessitates a solution that can effectively transform vast amounts of raw information into meaningful knowledge that can be comprehended by the human brain. Text summarization has emerged as a common technique in research to address the challenges posed by the enormous volume of data.

The analysis results provide an in-depth understanding of the ongoing research trends in text summarization. They offer references to publicly available datasets, discuss preprocessing techniques and employed features, and describe the commonly used techniques and methods employed by researchers for comparison and method development purposes. Moreover, this research paper concludes by highlighting several recommendations concerning the opportunities and challenges associated with text summarization research.

To further enhance the field of text summarization, it is crucial to explore approaches that can handle multi-document summarization, tackle bias and subjectivity, and preserve the original intent of the source text. Future research should also focus on refining existing methods, exploring innovative techniques, and leveraging advancements in natural language processing and machine learning algorithms. By addressing these challenges and embracing opportunities, the field of text summarization can continue to evolve and provide valuable solutions for information extraction and understanding.

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.NO.	WORD	FULL FORM
1	NLP	Natural Language Processing
2	INIT	Information Item
3	SVO	Subject-verb-object
4	LDA	Latent Dirichlet Allocation
5	NMF	Negative Matrix Factorization
6	TF-IDF	Term Frequency Inverse Document frequency
7	TF	Term Frequency
8	DF	Document Frequency
9	LSA	Latent Semantic Analysis
10	SVD	Singular Value Decomposition
11	NLTK	Natural Language Toolkit
12	RE	Regular Expression
13	BOW	Bag Of Words

CHAPTER 1

INTRODUCTION

With the rapid increase in web-based information available in various formats like text, video, and images, individuals often struggle to find relevant information of interest. When users query for information online, they are bombarded with numerous search results that may not be directly related to their specific needs. This information overload creates a time-consuming and effort-intensive task of sifting through documents to find the desired content. Automatic text summarization emerges as a valuable solution to address this problem.

Automatic summarization plays a crucial role in condensing source documents into concise and meaningful content that captures the main ideas without altering the information. By providing an effective summary, users can quickly grasp the key points without having to go through the entire document, ultimately saving time and effort. The text summarization process involves three steps: analysis, transformation, and synthesis. Analysis involves examining the source text and selecting relevant attributes. The transformation step processes the analyzed information, and finally, the synthesis step generates a summary representation.

Text summarization approaches are broadly categorized into extractive and abstractive summarization. Extractive summarization involves extracting important sentences or phrases from the source documents and grouping them together to create a summary while preserving the original text. On the other hand, abstractive summarization focuses on understanding the source text using linguistic methods to interpret and analyze its content. The goal of abstractive summarization is to generate a concise summary that conveys the information in a more generalized manner.

1.1 PROBLEM DESCRIPTION AND OVERVIEW

The internet offers a vast amount of information, but it has created a strange situation. People have access to so much data, yet they still crave wisdom. Keeping up with the billions of articles produced daily is challenging. To address this issue, we propose developing a Text Summarizer using specific methods.

1.2 OBJECTIVE

The aim of text summarization is to condense a document to its key points. This can be done by humans or using algorithms. The goal is to create a brief summary that captures the main ideas of the original text. Text summarization saves time when reading long papers, such as research articles, by providing a concise summary without leaving out important information.

CHAPTER 2

LITERATURE SURVEY

2.1 AUTOMATIC TEXT SUMMARIZATION

Authors: Ujjwal Rani, Karambir Bidhan

Content summarization involves condensing a source document into a concise form that captures the main ideas. There are two approaches to text summarization: abstractive and extractive. Abstractive summarization uses natural language processing techniques to generate summaries, while extractive summarization relies on statistical, linguistic, and heuristic methods to select and rank sentences. Various methods have been developed for text summarization in different languages. This paper focuses on exploring abstractive summarization techniques.

Abstractive Summarization

The goal is to create a brief summary that captures the main ideas of the source text. In abstractive summarization, new phrases and sentences can be included in the summary that are not present in the original text. Abstractive summarization approaches are rooted in a structure-based approach.

- **Structure Based Approach**

Structure-based methods in summarization involve utilizing prior knowledge and mental constructs such as templates, extraction rules, and alternative structures like trees, headings, and graphs to capture the most important information.

- **Tree based**

The tree-based approach in summarization involves using a dependency tree to analyze the content of a document. This method relies on language generators to generate accurate summaries. In the first step, the approach creates a dependency tree by dividing sentences into chunks. Then, it determines the importance of each part of the tree. Subtrees from different sections are identified and added to the dependency tree to enhance its structure. Finally, predefined elements are removed from the tree to refine the summary.

- **Rule based**

Information extraction rules produce output, and the content selection module selects the best option based on categorization to provide a suitable response. The primary objective of this method is to generate a summary that is more informative than the existing one.

- **Lead and body phrase method**

The lead and body phrase method focuses on identifying common phrases in the lead and body sections of the text. It then generates a summary by revising sentences through the insertion and substitution of these phrases. This approach is inspired by systems that combine sentences. However, a limitation of this method is that parsing errors can negatively impact the structure of the summary sentences, resulting in grammatical errors and redundancy.

- **Ontology based**

Ontology-based summarization is a new area within information extraction. It is inspired by the concept of text summarization in natural language processing. In ontology summarization, the goal is to extract relevant knowledge from an ontology and create a condensed version tailored to a specific user or task.

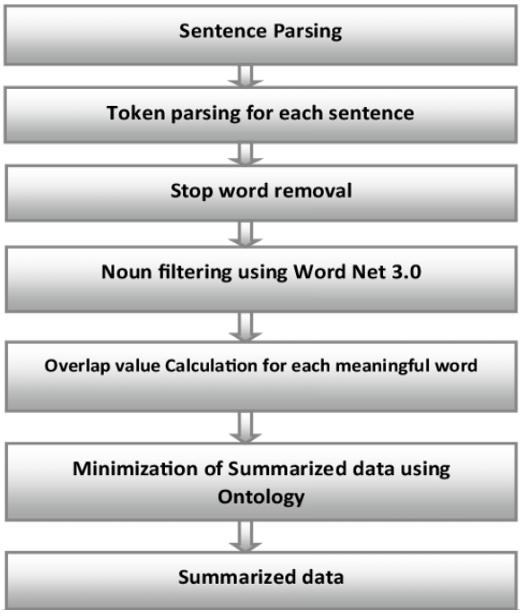


Fig 2.1 Ontology Based Summarization

- **Graph based**

Text summarization can be achieved using a graph-based approach, specifically through an unsupervised technique. In this technique, sentences or words are ranked based on their position within a graph. The primary objective of the graph-based method is to identify the most important sentences within a document. The graph represents the document or sentences as nodes, while edges connect nodes that share common information. The weightage or importance of each sentence is determined by assigning initial weights to the nodes of the graph.

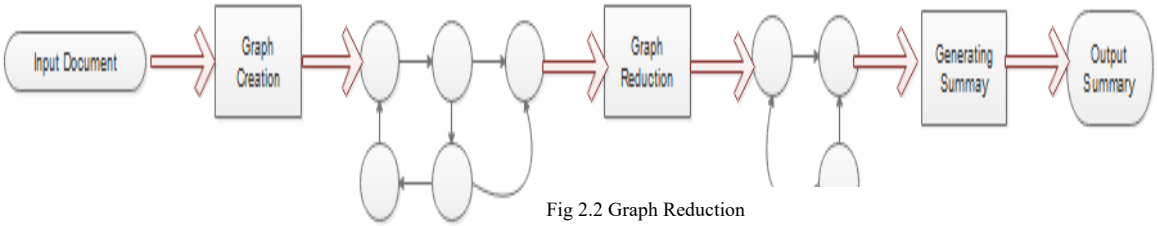


Fig 2.2 Graph Reduction

- **Information item (INIT) based**

This technique utilizes abstract representation to generate a precise summary of the document, rather than directly using sentences from the source document. The fundamental unit of abstract representation is the information item (INIT), which captures relevant information. The resulting summary is concise, intelligent, informative, and avoids excessive repetition. At the initial stage, a parser assists in the syntactical analysis of the document by retrieving INIT and forming subject-verb-object (SVO) structures. A language generator is employed to produce sentences. The sentences are then ranked based on their frequency scores, and only the high-ranking sentences are included in the summary. However, this approach is prone to grammatical errors and may struggle with generating meaningful sentences. Furthermore, the linguistic quality of the summary produced is often low due to incorrect parses.

2.2 NATURAL LANGUAGE PROCESSING (NLP) BASED TEXT SUMMARIZATION

Author: Ishitva Awasthi, Kuntal Gupta, Prabjot Singh Bhogal

In practical terms, Natural Language Processing (NLP) involves the use of computers to perform various tasks, including text summarization, parsing, entity recognition, and speech recognition. Text summarization in NLP is commonly achieved by extracting the main paragraphs or key information from a given text.

Text summarization is a process within NLP where artificial intelligence or machine-learning algorithms analyze text to infer its main topic or subject. Unlike deep learning, text summarization is focused on machine-based approaches, which are well-suited for generating quick outputs when dealing with short-text processing requirements.

Text summarization using NLP is a fundamental process for condensing text content by highlighting key features and presenting them in a concise manner. NLP, in general, refers to the use of computer systems to enhance the interaction between machines and humans through natural language understanding and processing.

- **Supervised Text Summarization:**

Supervised text summarization relies on having labeled training data, where each document is accompanied by its corresponding summary. This method involves training a machine learning model, such as a sequence-to-sequence model, using this labeled data. The model is trained to understand the relationship between the input document and its corresponding summary by learning from the provided examples. During training, the model is exposed to both the original document and the desired summary, allowing it to learn how to generate concise summaries by predicting the most suitable sequence of words. However, this approach necessitates a significant amount of annotated data, and the quality of the generated summaries depends on the quality and representativeness of the training data used.

- **Unsupervised Text Summarization:**

Unsupervised text summarization, in contrast to supervised methods, does not depend on labeled data. Its objective is to automatically extract the most essential information from

the source document without predefined summaries. There are various approaches used in unsupervised summarization:

Fig 2.3. Supervised & unsupervised summarization

2.3 EXTRACTIVE TEXT SUMMARIZATION

AUTHORS: Arpita Sahoo, Dr.Ajit Kumar Nayak

- **Summarization attributes of extractive**

In extractive text summarization, the focus is on identifying relevant sentences from the original document and including them in the summary. This approach involves analyzing the document to determine the importance of each sentence based on certain features or criteria. The sentences that meet the criteria are then selected and combined to form the summary. This method aims to extract and present the most crucial information from the source document without modifying or rephrasing the sentences.

- **Cue words features:**

Cue words or clauses are groups of words that appear around key words such as "summary," "reflects," "concludes," "purpose," "because," and so on. These cue words indicate the overall content of the document and can serve as indicators for determining which sentences should be included in the summary. By identifying these cue words or clauses, one can gain insights into the main ideas and important information contained within the document, allowing for effective selection of sentences for the summary.

- **Keyword features:**

A crucial feature of extractive summarization is the keyword feature. In this approach, sentences that contain a significant number of keywords are considered for inclusion in the summary. Keywords are specific words or phrases that carry substantial meaning and represent important concepts or ideas within the document. By identifying and prioritizing sentences that contain a high concentration of keywords, extractive summarization aims to capture the essential information and convey it in a concise summary.

- **Title word feature:**

In extractive summarization, the sentences that include the title or title-related terms are regarded as important and necessary sentences. These sentences are considered significant because they directly relate to the main subject or topic of the document. Including these title-related sentences in the final summary helps ensure that the summary captures the core

essence of the document and provides a clear representation of its main focus. By incorporating these important sentences, the summary can effectively convey the key information to the readers.

- **Topic Modelling:**

Topic modeling techniques, like Latent Dirichlet Allocation (LDA) or Non-Negative Matrix Factorization (NMF), can be employed to uncover the latent topics or themes present in a document. These techniques help in identifying the underlying patterns and topics by analyzing the distribution of words and their relationships within the text. By applying topic modeling, the summarization process can be guided by identifying the most relevant and representative sentences for each topic. This approach enables the extraction of key information from different aspects of the document, resulting in a more comprehensive and informative summary.

- **Sentence Similarity**:

Assessing the similarity between sentences plays a crucial role in extractive summarization. It helps in determining the redundancy or similarity of sentences within a document, enabling the identification of repetitive information. By measuring the similarity between sentences, redundant or similar sentences can be identified and potentially excluded from the summary. This process helps in improving the overall quality of the summary by reducing redundancy and ensuring that each included sentence provides unique and valuable information.

- **Sentence length feature**:

In text summarization, the length of sentences is an important consideration. The goal is to reduce the length of sentences while preserving their content and meaning. This is done to create concise summaries that capture the essential information from the original document. By condensing the sentences without altering their intended message, the summarization method aims to provide a shorter version of the text that is easier to comprehend and digest.

- **Upper case word feature**

In the process of text summarization, it can be beneficial to include words that are written in all uppercase or are considered abbreviations. These words often carry specific significance or represent key concepts in the document. By including them in the summary, the

summarization method ensures that important information and specific terminology are retained, allowing the summary to accurately convey the main ideas of the original document.

- **Term frequency:**

In text summarization, the TF-IDF (Term Frequency-Inverse Document Frequency) method can be employed to calculate the frequency of each word within sentences. This technique assigns a higher weight to words that have a higher frequency within a particular sentence and are relatively rare across the entire document collection. By considering the TF-IDF scores, the summarization method can prioritize words that occur frequently within sentences and are potentially more significant in conveying the overall meaning of the document. Including such high-frequency words in the summary can help ensure that the most important and representative information is captured.

- **Limitation of extractive method**
    - **Redundancy**

Extractive summarization methods indeed focus on selecting important and relevant sentences, but redundancy can be a challenge. It is possible that similar information is repeated across multiple selected sentences, which can hinder the overall conciseness of the summary and may not effectively capture the most essential information.

To address this issue, various techniques can be employed. One approach is to assess the similarity between selected sentences and identify redundant information. By measuring the similarity, redundant sentences can be identified and potentially removed to enhance the overall coherence and conciseness of the summary. Additionally, incorporating sentence compression or fusion techniques can help condense redundant information while preserving the key ideas.

Overall, minimizing redundancy in extractive summarization is crucial to create concise and informative summaries that efficiently capture the main points of the source document.

- **Lack of Coherence**

One of the challenges in extractive summarization is maintaining the coherence and flow of the summary. Since extractive methods rely on selecting and stitching together existing sentences, there can be a lack of smooth transitions between the sentences, resulting in a summary that may not read as a cohesive text.

To address this issue, additional post-processing steps can be applied to improve the coherence of the summary. Techniques such as sentence reordering, rephrasing, and using connecting phrases or discourse markers can help improve the overall flow and readability of the summary. Additionally, incorporating language generation models or abstractive methods can be explored to generate more coherent and cohesive summaries by paraphrasing and rephrasing the extracted sentences.

It is worth noting that achieving perfect coherence in extractive summarization can be challenging, as the method heavily relies on the available sentences. However, by employing various post-processing techniques and considering alternative summarization approaches, the coherence of the summary can be improved to provide a more cohesive reading experience.

- **Information Loss**:

When using extractive summarization, important details can be missed if they are spread across multiple sentences or not included in the selected sentences. This means that the summary may not fully capture the complete context or subtle details present in the original text.

- **Sensitivity to Sentence Order**

In extractive summarization, sentences are often chosen individually without considering their overall order in the summary.

- **Limited Adaptability to Diverse Texts**:

Extractive methods may face difficulties when dealing with complex texts like scientific papers, legal documents, or highly technical writings. The focus on analyzing and ranking sentences at the individual level may not accurately capture the main ideas or concepts present in such texts..

2.4 TEXT SUMMARIZATION: AN OVERVIEW

Author: Mr.S.A.Babar

- **Main steps for text summarization**:

The process of summarizing documents typically involves three main steps: identifying the topic, interpreting the content, and generating the summary.

- **Topic Identification:**

To identify the most important information in the text, various techniques are employed, such as analyzing the position of phrases, identifying cue phrases, and examining word frequency. Among these techniques, those based on the position of phrases tend to be the most effective for topic identification.

- **Interpretation:**

In the interpretation step of abstract summaries, various subjects or topics are combined to create a cohesive and overarching content. This process involves integrating different aspects or themes from the original text to form a comprehensive and generalized summary.

- **Summary Generation:**

In the interpretation step, the system employs a text generation method. This technique involves generating new text based on the combined subjects and themes identified in the previous step. The system uses algorithms or models to generate coherent and meaningful sentences that capture the essence of the original text in a concise manner.

- **Text summarization history:**

Extractive summarization methods primarily rely on scoring sentences within the source document. During the 1970s, researchers introduced frequency-based methods as a statistical approach to text summarization. These methods involved analyzing the frequency of important terms within sentences to determine their importance. Early systems such as AutoSummarize and Coh-Metrix were developed during this time, utilizing term frequency analysis to identify significant sentences for summarization.

- **Methods of extractive summerization**
    - **TF-IDF**

TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a technique used to measure the relevance of a word in a text document. It calculates a numerical value that increases as the number of times a word appears in the text increases. TF-IDF takes into account both the frequency of a word within a specific document (term frequency) and its importance in the overall corpus of documents (inverse document frequency). This approach helps identify words that are more significant and informative within a particular document compared to others.

Terminologies:

- **Term Frequency:**

In a document (d), the frequency (tf) of a word (t) represents the number of times that word appears in the document. The formula for calculating the term frequency is the count of the word (t) in the document (d) divided by the total number of words in the document (d). The term frequency helps indicate the significance or relevance of a word within a specific document. It does not consider the order of the words in the document; it focuses solely on the occurrence and frequency of the word.

tf(t,d) = count of t in d / number of words in d

- **Document Frequency**

This test is very similar to TF (Term Frequency) but with one difference. In the entire collection of documents (corpus), DF (Document Frequency) represents the number of occurrences of a term (t) in the document set (N), whereas TF is the frequency count of the term (t) in a specific document (d). DF provides information about how commonly a term appears across the entire document set, whereas TF focuses on the frequency of a term within a single document.

df(t) = occurrence of t in documents

2.5 TEXT SUMMARIZATION TECHNIQUES

Author: Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi

- **Latent Semantic Analysis (LSA)**

Latent Semantic Analysis (LSA) is a statistical technique used in text analysis and natural language processing. It can also be applied to tasks involving text summarization. LSA works by analyzing the patterns of word co-occurrence in order to uncover underlying semantic connections between words and documents. By doing so, it aims to capture the latent or hidden meaning present in the text data.

Benefits of LSA

LSA offers several benefits and applications in text analysis:

1. Dimensionality Reduction: LSA helps in reducing the dimensionality of large text-based datasets. By representing documents and words in a lower-dimensional space, it simplifies the analysis and makes it more computationally efficient.

2. Topic Understanding: LSA provides insights into the latent topics encoded within a text corpus. By identifying patterns and associations between words and documents, it helps in understanding the underlying themes or topics present in the data.

3. Word Association Analysis: LSA allows for the examination of word associations within a text corpus. It can uncover relationships and connections between words, providing valuable information about how words are semantically related.

4. Term Relations: LSA can identify relations between terms by analyzing their co-occurrence patterns. It helps in understanding the semantic relationships and connections between terms, enabling better information retrieval and retrieval of related terms.

5. Prior Art Searches: LSA has been utilized in assisting prior art searches for patents. By analyzing the relationships between terms and documents, it helps in identifying relevant prior art and determining the novelty of inventions.

Overall, LSA is a versatile technique that aids in dimensionality reduction, topic understanding, word association analysis, term relations, and assisting in prior art searches for patents.

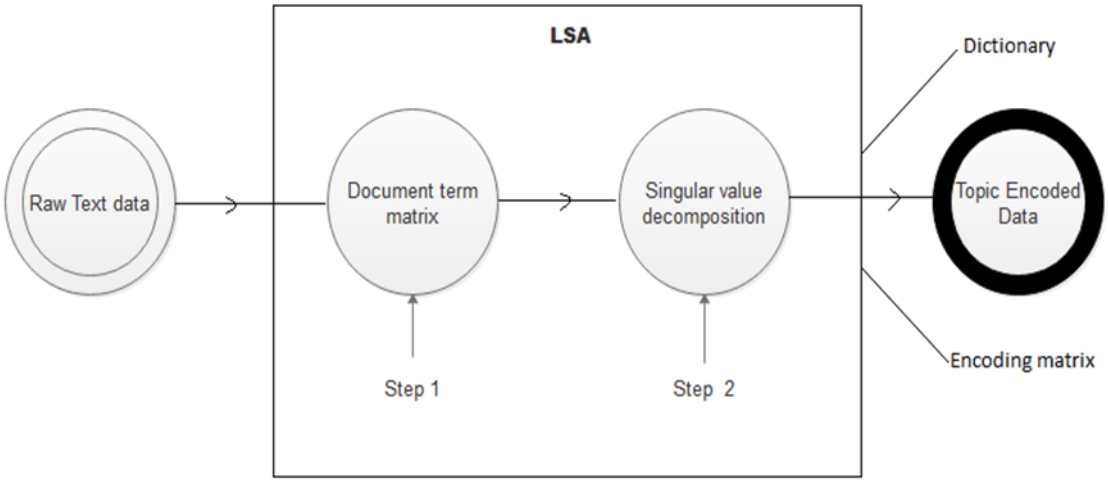


Fig 2.4 LSA Processing

- **Limitations**

1. Lack of Word Order and Syntactic Information: LSA overlooks the information related to word order, syntactic relationships, and morphologies present in the text. This can be a limitation because these aspects play a crucial role in understanding the meaning of words and texts. Without considering these elements, LSA may not capture the full linguistic nuances and context.

2. Limited Use of World Knowledge: LSA solely relies on the information available within the input documents and does not incorporate external world knowledge. This can be a drawback as incorporating external knowledge, such as background information or domain-specific knowledge, can enhance the accuracy and contextual understanding of the analysis.

3. Performance Challenges with Large and Inhomogeneous Data: LSA's performance tends to decline when dealing with larger and more diverse datasets. This decline in performance is mainly attributed to the Singular Value Decomposition (SVD) algorithm, which is computationally intensive and can struggle with complex and heterogeneous data. As a result, the accuracy and efficiency of LSA may decrease as the data size and heterogeneity increase.

In summary, the limitations of LSA include the neglect of word order and syntactic information, the limited utilization of external world knowledge, and challenges in handling larger and more diverse datasets that can impact the algorithm's performance.

5.6 AUTOMATIC MULTIPLE DOCUMENTS TEXT SUMMARIZATION

Author: Md. Majharul Haque1, Suraiya Pervin1, and Zerina Begum2

- **Time based method**

  A time-based approach to text summarization focuses on understanding the temporal aspects of information in a document and creating a summary based on it. The goal is to either gather the most important and relevant information from different time periods or generate a summary that emphasizes current or up-to-date information. Here is a simplified explanation of how a time-based text summarization technique could work:

  1. Preparing the Document: The document is processed to remove unnecessary elements like stop words or punctuation. It is also divided into sentences or smaller parts for analysis.

  2. Extracting Timestamps: If the document contains timestamps or date information, they are extracted using techniques like regular expressions or natural language processing.

  3. Selecting a Time Window: A specific time window is chosen to define the period of interest for the summary. This window can be fixed, such as the last month or year, or dynamically determined based on the timestamps in the document.

  4. Scoring Relevance: Each sentence or unit of text is assigned a score based on its relevance to the selected time window. Sentences that fall within the time window or have timestamps close to it are considered more relevant. Different algorithms, like keyword matching or semantic similarity, can be used for this scoring.

  5. Ranking Importance: In addition to relevance, sentences can be ranked based on their importance or significance within the time window. Factors like the number of occurrences, popularity, or source credibility may determine the importance ranking.

6. Generating the Summary: The top-ranked and most relevant sentences from the time window are chosen to create the summary. The length of the summary can be predetermined or adjusted based on the available content and desired level of detail.

7. Post-processing: The generated summary may undergo further refinement to improve coherence and readability. Techniques like sentence compression, removing redundant information, or applying linguistic rules can be used for post-processing.

A time-based text summarization approach involves analyzing the temporal aspects of information, selecting relevant sentences from a specific time window, and generating a concise summary that captures important details.

5.7 IMPROVING PERFORMANCE OF TEXT SUMMARIZATION

Author: S.A.Babara,Pallavi D.Patilb

- **Fuzzy logic scoring**

  In text summarization, fuzzy logic scoring is a method that applies principles and techniques of fuzzy logic to assign scores or weights to sentences for determining their importance or relevance in the summary. Fuzzy logic allows for the consideration of different levels of truth and membership in linguistic variables, which helps in dealing with the inherent ambiguity and imprecision present in natural language. The process of using fuzzy logic scoring in text summarization involves the following steps:

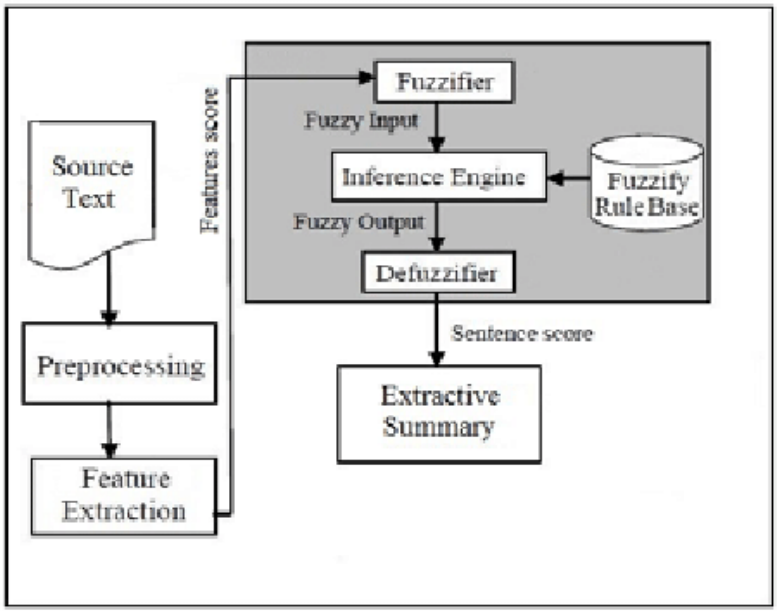


Fig 2.5 Fuzzy Logic Diagram

5.8 AUTOMATED TEXT SUMMARIZATION

Author: Eduard Hovy and ChinYew Lin

- **Cluster based method**

In this approach, the content of a document is analyzed by extracting and representing its semantic structure using triplets consisting of subjects, verbs, and objects associated with each sentence. These triplets are then clustered based on their similarity, grouping together those with similar information. The triplets statements serve as the fundamental units in the summarization process. By identifying and clustering similar triplets, redundant information can be minimized. This allows for the construction of a summary using a sequence of sentences that are related to the computed clusters.

Cluster 1	⟨ambulance, be, scene⟩ ⟨police, be, scene⟩ ⟨ambulance, stream, area⟩ ⟨people, be, street⟩ ⟨pedestrian, peer, sky⟩
Cluster 2	⟨minister, inform, crash⟩ ⟨president, inform, incident⟩ ⟨spokesman, say, minister⟩
Cluster 3	⟨airplane,crash,tower⟩ ⟨plane, strike, building⟩ ⟨clock, fall, floor⟩ ⟨terror, attack, Washington⟩ ⟨terror, attack, New York⟩

Fig 2.6 Clusters of Triplets

- **MACHINE LEARNING**
  - A training documents set along with its summaries in extractive form are given as input to the training stage.
  - Machine learning approach classified as supervised, unsupervised or semi-supervised.
  - Bayesian rule is used to statistically determine the classification probabilities

P (s∈<S | F1, F2, ..., FN)= P (F1, F2, ..., FN | s∈S) *P (s∈S) / P (F1, F2,...,FN)

In this framework, each sentence S from the document collection is considered, and a set of features F1, F2,...,FN is used for classification. The goal is to determine whether each sentence should be included in the summary or not. The summary S to be produced is a collection of sentences that are deemed to be summary sentences based on the given features. The probability P(s ∈ <S | F1, F2,...,FN) represents the likelihood that a sentence s belongs to the summary based on the observed features. By calculating these probabilities, sentences can be classified as either summary sentences or non-summary sentences, allowing for the construction of an effective summary.

5.9 AUTOMATIC TEXT SUMMARIZATION: A COMPREHENSIVE SURVEY

Author: Wafaa S. El-Kassas

- **Query based method**

The query-based text summarizer utilizes a graph structure to analyze the relationships between sentences and words. It leverages various methods and algorithms that incorporate statistical and linguistic techniques. In the past, different approaches have been used individually, but to enhance the effectiveness of the summarizer, a combination of these techniques is necessary. By integrating multiple approaches, the summarizer can achieve improved efficiency and generate more accurate and informative summaries.

Algorithm:

1. Arrange sentences based on their score.
2. Include sentences from the title or topic of the document.
3. Include the first-level heading as part of the summary.
4. Check if the summary size limit is not exceeded.
5. Add the highest scored statement to the summary.
6. Include the structural context of the statement if it has not already been added.
7. Add the highest-level heading above the extracted text (let's refer to this heading as 'h').
8. Add the heading preceding 'h' at the same level.
9. Add the heading following 'h' at the same level.
10. Repeat steps 7, 8, and 9 for the next highest-level headings.
11. Continue these steps until the while loop is terminated.

This algorithm outlines the process of generating a summary by arranging sentences based on their scores and incorporating relevant headings and structural context. The loop continues until the summary size limit is reached or no further headings are available.

2.10 A SURVEY OF TEXT SUMMARIZATION EXTRACTIVE TECHNIQUES

Author: Vishal Gupta

- **Neural Network**
  - A neural network is a network of interconnected artificial neurons that utilize a numerical model for processing data.
  - In the context of text summarization, the approach involves training the neural network to recognize the type of sentences that should be included in the summary.
  - The neural network is trained using sentences from a test paragraph, where each sentence is evaluated to determine if it should be included in the summary or not.
  - Once the neural network is trained, small weights are pruned to remove less important features.
  - The summary is then generated by selecting sentences with high scores based on the neural network's evaluation.

- **Multi-document extractive summarization**

Multi-document extractive summarization is a technique used to create concise summaries by extracting important phrases or sections from multiple source documents. Unlike single-document summarization, which focuses on condensing information within a single document, multi-document summarization takes into account information from a collection of related papers. This approach aims to provide a comprehensive overview by considering multiple perspectives and sources of information. By selecting key phrases or sections from various sources, multi-document summarization helps to capture the essential information and present it in a concise and informative manner.

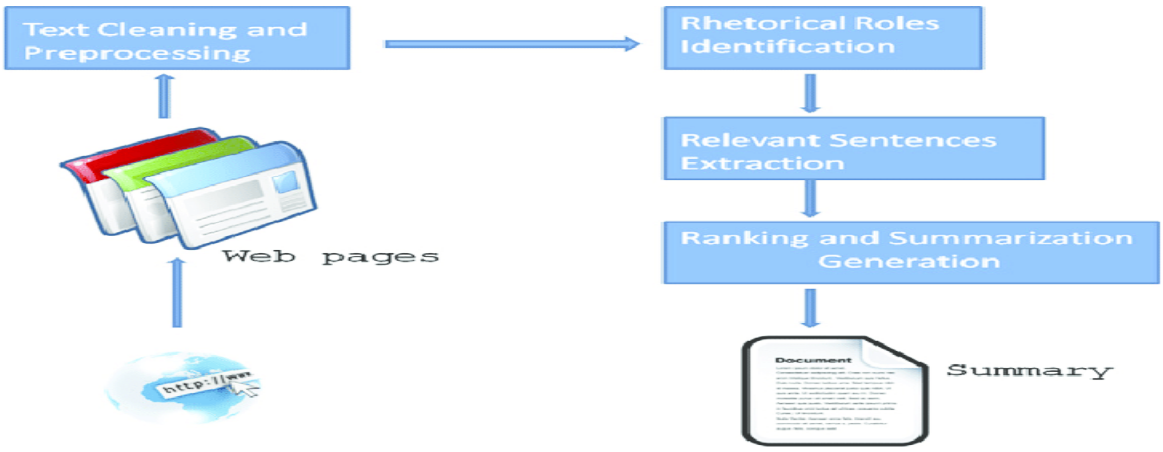


Fig 2.7 Multi document Summarization

CHAPTER 3

PROPOSED SYSTEM

3.1) SYSTEM SPECIFICATIONS & REQUIREMENTS

3.1.1) NUMPY

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project.

3.1.2) PANDA

Panda is a python library for data analysis. It was started by Wes McKinney in 2008.It has functions for analyzing, cleaning, exploring, and manipulating data. The name “Pandas” has a reference to both “Panel Data”, and “Python Data Analysis” .Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values.

3.1.3) NLTK

The Natural Language Toolkit or NLTK, is one of the premier libraries for developing Natural Language Processing (NLP) models. It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

3.1.4) PYTHON

Python is an interpreted, object-oriented & high-level programming language developed by Guido van Rossum. It was originally released in 1991.Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn’t specialized for any specific problems. Python runs on an interpreter system, meaning that code can be executed as soon as it is written.

3.1.5) VISUAL STUDIO CODE

Visual studio code, also commonly referred to as a VS Code, is a source-code editor made by Microsoft with the Electron framework, for windows, Linux and macOS. Visual Studio Code is a free coding editor that helps us to start coding quickly. Any programming language can be coded in VS code. Visual Studio Code has support for many languages including python, java , C++, Javascript, and more.

3.1.6) RE (REGULAR EXPRESSION)

The Python "re" module provides regular expression support. In Python a regular expression search is written as: match = re. search(pat, str) The re.search() method takes a regular expression pattern and a string and searches for that pattern within the string. The’ re’ package is a built-in module in Python that provides functions for working with regular expressions, which are powerful tools for pattern matching and text manipulation.

3.1.7) WARNING

The "warnings" module provides a way to manage and control warning messages that are issued during the execution of a Python program. In NLP, the "warnings" package can be useful when working with external libraries or APIs. By using the "warnings" package, you can control how these warning messages are handled, whether to display them, ignore them, or raise an exception.

3.1.8) WORD_TOKENIZE

word_tokenize is a function in Python that splits a given sentence into words using the NLTK library. It is a common pre-processing step in NLP tasks where text data needs to be processed at the word level. One popular tool used for word tokenization in NLP is the Natural Language Toolkit (NLTK), which is a library for Python. NLTK provides a method called ' word_tokenize()' that tokenizes a text string into a list of words.

3.1.9) TEXTBLOB

TextBlob is a popular Python library for natural language processing (NLP). It provides a simple and intuitive API for common NLP tasks such as noun phrase extraction, sentiment analysis, language translation, and more. TextBlob is built on top of NLTK (Natural Language Toolkit) and offers an easy-to-use interface for NLP operations.

3.1.10) WORDCLOUD

Word clouds are a visualization technique commonly used in natural language processing (NLP) to display the most frequent words or terms in a text corpus. It is great for visualizing unstructured text data and getting insights on trends and patterns.

3.1.11) STOPWORDS

Stop words are a set of commonly used words in any language. For example, in English, "the", "is", & "and", would easily qualify as stop words. In NLP, stop words are used to eliminate unimportant words, allowing applications to focus on the important words instead.

3.2) ALGORITHM

1. Begin
2. Import necessary libraries:
   2.1.numpy (as np): For numerical operations
   2.2.pandas (as pd): For data manipulation
   2.3.warnings: For handling warnings
   2.4.re: For regular expression operations
   2.5.nltk: Natural Language Toolkit for text processing
   2.6.nltk.tokenize: For tokenization
   2.7.nltk.sent_tokenize: For sentence tokenization
   2.8.textblob: For text processing and sentiment analysis
   2.9.string: For string operations
   2.10.nltk.corpus.stopwords: For stopwords removal
   2.11.statistics.mean: For calculating the mean
   2.12.heapq.nlargest: For finding the largest elements in a list

2.13.wordcloud: For creating word clouds
3. Define 'stop words' & 'punctuations' for initializing set of stopwords and punctuations.
4. Define 'warning' module to ignore warning messages.
5. Reads three CSV files 'articles1.csv,'articles2.csv' & 'articles3.csv into three pandas dataframes 'df_1','df_2' & 'df_3'. Compares the column names of 'df_1' & 'df_2' using '==' operator
5.1.Repeat the same for 'df_2' & 'df_3'
5.2.Creates a list 'd' containing 'df_1','df_2' & 'df_3'
5.3.Concatenates the DataFrames in the list 'd' using 'pd.concat(d,keys==['x','y','z'])'
5.4.Rename the column 'content' to 'article' in the concatenated dataframe
5.5.Prints the first few rows of the concatenated DataFrame using 'df.head()'.
5.6.Prints the shape of the concatenated DataFrame using 'df.shape'
5.7.Drop the 'Unnamed: 0' column from the concatenated dataframe
5.8.Print the first few rows of the modified DataFrame.
6. import 'seaborn' and 'matplotlib' libraries for data visualization.
6.1.It aims to create two count plots to visualize the distribution of publications and articles according to the year.
6.2.Set the figure size for the plots.
6.3.Set the font scale and style for seaborn.
6.4.Create the first count plot for the distribution of publications.
6.5.Rotate the x-axis labels by 45 degrees for better readability.
6.6.Set the labels and title for the first plot.
6.7.Replace a specific value in the 'year' column of the DataFrame 'df'
6.8.Create the second count plot for the distribution of articles by year
6.9.Set the labels and title for the second plot.
7. Analyse a dataframe 'df' that contains the column 'author'
7.1. calculate the frequency count of each unique value in the 'author' column of the DataFrame 'df'.
7.2. sets the font scale, style, and grid style for the seaborn library using 'sns.set(font_scale=1,style='whitegrid')'
7.3. selects the top 80 most frequent authors from the DataFrame using 'df_author=df.author.value_counts().head(80)'.
7.4. create a 'barplot' function to create a horizontal barplot. The x-axis represents the count of authors, and the y-axis represents the author names.
7.5. Add labels and titles to the x and y axis
7.6. Adjust the plot appearance using various functions
7.7. Replace the contractions
7.8. Define 'contractions_dict', it maps certain contractions to their expanded forms.
8. import the 're' module for regular expressions.
8.1.Compile a regular expression pattern ' contractions_re' using 're.compile'.
8.2. Define a function name called 'cleanhtml' that takes a single parameter 'raw_HTML'
8.3. Compile another regular expression pattern 'cleanr' to match and remove HTML tags using the '<.*?>' pattern
8.4. use 're.sub' to substitute the matches of 'cleanr' in 'raw_ html' with an empty string, effectively removing the HTML tags.

8.5. store the result in 'cleantext'
8.6. return the cleaned text as output of the function.
9. Define a function named 'expand_contractions' that takes two parameters:'s' & 'contractions_ dict'.
9.1. Define an inner function named 'replace' that takes a single parameter 'match'.
9.2. inside 'replace' return the expansion of the matched contraction 'match.group(0)' from the ' contractions_dist' .
9.3.Define a function named 'preprocessing' that takes a single parameter 'article'.
9.4. converted the article to lowercase
9.5. remove the HTML tags using ' cleanhtml'
9.6. use regular expression 're' for removing email addressses & URLs
9.7.Replace the non-breaking space character with a regular space character.
9.8. expand the contractions in the article using 'expand_contractions'.
9.9. Return the preprocessed 'article' as output of the function.
10.Apply a lambda function to the 'article' series for removing possessive forms of words and reducing consecutive spaces.
10.1. Extends the preprocessing steps for the 'article' data by removing punctuation and stop words.
10.2. It applies lambda functions and regular expressions to achieve these modifications.
10.3. Store the result back in the ' article' series.
11. Analyze the frequency and score of sentences in an article using 'word_frequency' & 'sentence_score'.
12.Using 'summary(sentence_score_Owo)',it calculates the number of sentences to include in the summary by taking 25% of the total number of sentences. It selects the top-scoring sentences using the 'nlargest' function and appends them as a summary to the 'summary list'.
12.1.calculates the word frequencies using the ' word_frequency' function.
12.2.calculates sentence scores using the 'sent_token' function.
12.3.Generates the summarized article using the 'summary' function.
12.4.The last part of the code calls the 'article_summarize' function with a subset of article dataframe.
12.5. Then it prints the actual length of the first article, displays the full article text, prints the length of the summarized article, displays the summarized article, and finally calls the 'word_cloud' function to generate and display the word cloud visualization.
13.1) Import the necessary libraries:
13.1.1) 'newspaper'-library for scraping articles from websites.
13.1.2) 'reportlab'-library for generating PDF documents.
13.1.3) 'sumy'-library for text summarization.
13.2) Import specific modules from the libraries:
13. 2.1) 'article' class from 'newspaper' library to extract information from newspaper articles.
13.2.2) 'SimpleDocTemplate' , 'paragraph' and 'spacer' classes from 'reportlab.lib.styles' module for creating PDF documents.
13.2.3) 'getSampleStyleSheet' function from 'reportlab.lib.styles' module for defining document styles.

13. 2.4) 'plaintextParser', 'Tokenizer' & 'LsaSummarizer' classes from the 'sumy.parsers.plaintext' & 'sumy.nlp.tokenizers' & 'sumy.summarizers.lsa' modules respectively for text summarization.
13.3) Perform text summarization by initialising an newspaper article.
13.3.1) use a 'parse()' method to extract the article's information.
13.3.2) initialise a 'LsaSummariser' object.
13.3.3) use the 'analyze()' method of the 'LsaSummarizer' to compute the summary.
13.3.4) Get the summarized sentences using the 'get_summary()' method.
13.4) Generate a PDF document by initialising 'SimpleDocTemplate' object with the desired output file name and page size.
13. 4.1) Create a list of paragraphs and spacers with the summarized sentences and styles.
13.4.2) Build the document using the 'build()' method of the 'SimpleDocTemplate' object.
14) End.

CHAPTER 4

IMPLEMENTATION

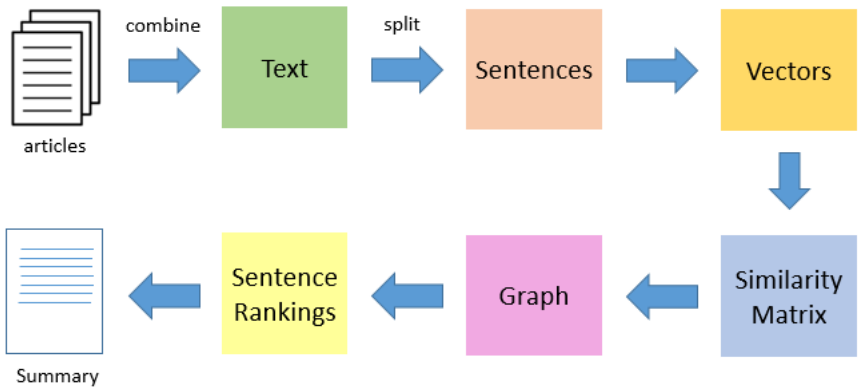


Fig 4.1 Implementation Steps

4.1 TOKENIZATION

In any NLP project, the first important step is text preprocessing. Preprocessing involves organizing the input text in a way that can be easily analyzed. There are different techniques for text preprocessing, such as

- removing stop words
- dividing the text into meaningful units called tokens
- reducing words to their base form (stemming).

Among these techniques, tokenization plays a crucial role. It breaks the text into words, sentences, or other meaningful elements called tokens.
Tokenization is essential because it allows us to convert unstructured text into numerical data that can be used for machine learning. These tokens can trigger specific actions or be used as features in a machine learning pipeline.

Tokenization can separate sentences (sentence tokenization) or words (word tokenization).

- Example of sentence tokenization

```
sent_tokenize('Life is a matter of choices, and every choice you make makes you.')
['Life is a matter of choices, and every choice you make makes you.']
```

Fig 4.2 Sentence Tokenization

- Example of word tokenization

```
word_tokenize("The sole meaning of life is to serve humanity")
['The', 'sole', 'meaning', 'of', 'life', 'is', 'to', 'serve', 'humanity']
```

Fig 4.3 Word Tokenization

- **some tools for tokenization**

Tokenization in Python may be simple, it's the foundation to create good models and help us to understand the text corpus. The few tools available for tokenizing text content like NLTK, TextBlob, spacy, Gensim, and Keras.

4.1.1 NLTK Word Tokenize

The NLTK (Natural Language Toolkit) is a Python library that provides tools and resources for Natural Language Processing tasks. It offers user-friendly interfaces for more than 50 corpora and lexical resources, including WordNet. NLTK also provides a collection of text processing libraries for tasks like classification, tokenization, stemming, and tagging.

To tokenize sentences and words in text using NLTK, you can utilize the tokenize module. First, you need to import the necessary functions from the NLTK library.

```
import nltk
from nltk.tokenize import (word_tokenize,
                           sent_tokenize,
                           TreebankWordTokenizer,
                           wordpunct_tokenize,
                           TweetTokenizer,
                           MWETokenizer)
text="Hope, is the only thing stronger than fear! #Hope #Amal.M"
```

Fig 4.4 importing packages for tokenization

4.1.2 Word and Sentence tokenizer

```
print(word_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal.M']
```

```
print(sent_tokenize(text))
```

```
['Hope, is the only thing stronger than fear!', '#Hope #Amal.M']
```

Fig 4.5 Word & Sentence Tokenizer

4.1.3Punctuation-based tokenizer

This tokenizer splits the sentences into words based on whitespaces and punctuations.

```
print(wordpunct_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal', '.', 'M']
```

Fig 4.6 Punctuation based Tokenizer

4.1.4 TextBlob Word Tokenize

TextBlob is a Python library designed for working with textual data. It offers a unified and convenient API for various natural language processing (NLP) tasks, including part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

To perform word tokenization using the TextBlob library, you can use the following code snippet:

```
from textblob import TextBlob
text= " But I'm glad you'll see me as I am. Above all, I wouldn't want people to think that I want to prove anything

blob_object = TextBlob(text)
# Word tokenization of the text
text_words = blob_object.words
# To see all tokens
print(text_words)
# To count the number of tokens
print(len(text_words))
```

```
['But', 'I', "'m", 'glad', 'you', "'ll", 'see', 'me', 'as', 'I', 'am', 'Above', 'all', 'I', 'would', "n't", 'want',
'people', 'to', 'think', 'that', 'I', 'want', 'to', 'prove', 'anything', 'I', 'do', "n't", 'want', 'to', 'prove', '
anything', 'I', 'just', 'want', 'to', 'live', 'to', 'cause', 'no', 'evil', 'to', 'anyone', 'but', 'myself', 'I', 'h
ave', 'that', 'right', 'have', "n't", 'I', 'Leo', 'Tolstoy']
55
```

Fig 4.7 TextBlob Tokenization

Indeed, the TextBlob tokenizer has certain built-in rules for tokenization. One of these rules is the removal of punctuation marks, which are considered separate tokens. This helps in separating words from punctuation symbols in the text.

Furthermore, TextBlob also has specific rules for handling contractions in English. Contractions such as "can't" or "won't" are treated as separate tokens, splitting them into their constituent parts. For example, "can't" would be tokenized into "can" and "'t".

These tokenization rules in TextBlob ensure that the text is divided into meaningful units, such as words or contractions, which can then be further processed or analyzed in NLP tasks.

4.2 WORD EMBEDDINGS

Word Embedding, also known as Word Vector, is a method used to represent words and documents in a numerical form. It assigns a numeric vector to each word, allowing words with similar meanings to have similar representations. These word vectors capture various features that relate to the words, such as age, sports, fitness, or employment.

The purpose of word embeddings is to reduce the dimensionality of word representations and capture the inter-word semantics. By using word embeddings, we can input these numeric representations into machine learning models for tasks like training or inference. Word embeddings also help in visualizing patterns of word usage within the training corpus.

Word embeddings differ from other methods like Bag of Words (BOW), Count Vectorizer, or TFIDF, which rely solely on word counts and do not preserve syntactical or semantic information. Word embeddings provide a solution to the limitations of these methods by capturing both syntactical and semantic information. They result in lower-dimensional vectors, avoid sparse matrices, and reduce the computational complexity during training.

To generate word vectors, we can use different approaches. One approach is to assign vectors to emoticons based on their frequent usage in specific contexts, with the contexts being the features. Similarly, we can create word vectors for different words based on given features. Words with similar vectors are likely to have similar meanings or convey similar sentiments.

word embeddings are a powerful tool for representing words in a numeric form, allowing us to work with text data in machine learning models while capturing semantic and syntactical information.

The two different approaches to get Word Embeddings are:

- Word2Vec:

In Word2Vec, each word is assigned a vector representation. There are two common approaches to initialize these vectors: random vector and one-hot vector.

A one-hot vector represents a word using a binary vector where only one element is set to 1, while all others are set to 0. For example, if there are 500 words in the corpus, the length of the one-hot vector would be 500. Each word in the corpus is then assigned a unique one-hot vector.

Once the vectors are assigned, the Word2Vec algorithm uses a sliding window approach. It iterates through the entire corpus, considering a specified window size. Within each window, the algorithm captures the co-occurrence patterns between words.

By training on the corpus and analyzing these co-occurrence patterns, Word2Vec learns to generate vector representations for words that capture their semantic relationships and similarities. This allows it to capture the meaning and context of words in a lower-dimensional space.

Overall, Word2Vec is a powerful technique for representing words as vectors and capturing their semantic relationships based on co-occurrence patterns in a given corpus.

4.3 REMOVE PUNCTUATIONS, SPECIAL CHARACTERS AND NUMBERS.

Pre-processing plays a crucial role in improving the accuracy of various natural language processing (NLP) tasks such as sentiment analysis, feedback classification, translation, summarization, and more. When dealing with text data in its natural form, such as sentences, comments, paragraphs, tweets, etc., it is essential to apply pre-processing techniques to enhance the effectiveness of machine learning algorithms.

Here, we explore some common pre-processing techniques using Python as the programming language. By applying pre-processing steps, we can clean up the input data and focus on the most significant words while removing those that contribute minimal to no value. This helps algorithms to better understand the underlying patterns and extract meaningful information. Let's delve into the NLP pre-processing techniques using Python to prepare our text data for further analysis or modelling.

4.3.1 Removing HTML Tags

In many cases, when working with unstructured text, such as data obtained through web or screen scraping, there is often a significant amount of irrelevant information or noise. One common component that adds little value to understanding and analyzing text is HTML tags. Therefore, it is important to remove these tags.

To accomplish this, we can utilize the BeautifulSoup library, which provides functionality for cleaning up HTML tags from text.

```
# imports
from bs4 import BeautifulSoup# function to remove HTML tags
def remove_html_tags(text):
    return BeautifulSoup(text, 'html.parser').get_text()# call
function
remove_html_tags( ‘<html> \
 <h1>Article Heading</h1> \
 <p>First sentence of some important article. And another one. And
then the last one</p></html>’)
```

Fig 4.8 Removing HTML Tags

Output:

```
' Article Heading First sentence of some important article. And
another one. And then the last one'
```

Fig 4.9 Output for removed HTML tags

4.3.2 Removing Accented Characters

Accented characters play a significant role in indicating emphasis or clarifying the meaning of a word during pronunciation or understanding. While their usage in English is relatively limited, there is a possibility of encountering accented characters/letters in a free text corpus. Words such as résumé, café, prótest, divorcé, coördinate, exposé, latté, etc., contain accents that may arise from default keyboard settings or typing styles. Handling accented characters becomes crucial, especially when focusing on the analysis of the English language. Therefore, it is necessary to convert and standardize these characters into ASCII characters. For example, converting é to e.

```
# imports
import unicodedata# function to remove accented characters
def remove_accented_chars(text):
    new_text = unicodedata.normalize('NFKD', text).encode('ascii',
'ignore').decode('utf-8', 'ignore')
    return new_text# call function
remove_accented_chars('Sómě Áccěntěd těxt. Some words such as
résumé, café, prótest, divorcé, coördinate, exposé, latté.')
```

Output:

```
'Some Accented text. Some words such as resume, cafe, protest,
divorce, coordinate, expose, latte.'
```

Fig 4.10 Removing Accented Characters

4.3.3 Expanding Contractions

Contractions are shortened versions of words or syllables created by removing specific letters from words and combining them. They are indicated in writing by an apostrophe to represent the missing letters. Contractions are commonly used in both written and spoken forms of the English language. Nowadays, many text editors automatically induce contractions. For example, 'do not' becomes 'don't', 'I would' becomes 'I'd', and 'you are' becomes 'you're'.

Converting contractions back to their expanded, original forms helps in standardizing the text. To remove contractions, the contractions library provides a standard set of contractions.

```
# imports
from contractions import CONTRACTION_MAP # from contractions.py
import re # function to expand contractions
def expand_contractions(text, map=CONTRACTION_MAP):
    pattern = re.compile('({})'.format('|'.join(map.keys())),
flags=re.IGNORECASE|re.DOTALL)
    def get_match(contraction):
        match = contraction.group(0)
        first_char = match[0]
        expanded = map.get(match) if map.get(match) else
map.get(match.lower())
        expanded = first_char+expanded[1:]
        return expanded     new_text = pattern.sub(get_match, text)
    new_text = re.sub("'", "", new_text)
    return new_text# call function
expand_contractions("Y'all i'd contractions you're expanded don't
think.")
```

Output:

```
'You all i would contractions you are expanded do not think.'
```

Another way can be:

```
# imports
from pycontractions import Contractions
cont = Contractions(kv_model=model)
cont.load_models()# function to expand contractions
def expand_contractions(text):
    text = list(cont.expand_texts([text], precise=True))[0]
    return text
```

Fig 4.11 Expanding Contractions

4.3.4 Removing Special Characters

Special characters, also known as non-alphanumeric characters, are commonly found in comments, references, currency numbers, and other contexts. These characters do not

contribute to the understanding of text and can introduce noise into algorithms. Fortunately, we can use regular expressions (regex) to remove these characters and numbers from the text.

```
# imports
import re# function to remove special characters
def remove_special_characters(text):
    # define the pattern to keep
    pat = r'[^a-zA-z0-9.,!?/:;\"\'\s]'
    return re.sub(pat, '', text)

# call function
remove_special_characters("007 Not sure@ if this % was #fun! 558923
What do# you think** of it.? $500USD!")
```

Output:

```
'007 Not sure if this  was fun! 558923 What do you think of it.?
500USD!'
```

Fig 4.12 Removing Special characters

4.3.5 Removing Numbers

In addition to special characters, numbers can also be removed from text during text processing. Numbers may not contribute much information to text analysis, especially when dealing with textual data. Regular expressions (regex) can be used to eliminate numbers from the text. This step can be combined with the previous step of removing special characters, allowing both processes to be performed in a single step.

```
# imports
import re# function to remove numbers
def remove_numbers(text):
    # define the pattern to keep
    pattern = r'[^a-zA-z.,!?/:;\"\'\s]'
    return re.sub(pattern, '', text)

# call function
remove_numbers("007 Not sure@ if this % was #fun! 558923 What do#
you think** of it.? $500USD!")
```

Output:

```
' Not sure if this  was fun!  What do you think of it.? USD!'
```

Fig 4.13 Removing Numbers

4.3.6 Removing Punctuation

To remove punctuation from text, it can be combined with the step of removing special characters. One way to accomplish this is by utilizing the `string.punctuation` module, which provides a list of all punctuation characters. By keeping only the characters that are not in this list, we can effectively remove punctuation from the text.

```
# imports
import string# function to remove punctuation
def remove_punctuation(text):
    text = ''.join([c for c in text if c not in
string.punctuation])
    return text# call function
remove_punctuation('Article: @First sentence of some, {important}
article having lot of ~ punctuations. And another one;!')
```

Output:

```
'Article First sentence of some important article having lot of
punctuations And another one'
```

Fig 4.14 Removing Punctuations

4.3.7 Stemming

Stemming is a technique used to reduce inflected or derived words to their base or root form. It involves removing common word endings such as 's', 'es', 'ed', 'ing', 'ly', and so on. While the resulting stem may not always be an exact match to the original word, stemming helps to standardize text by reducing words to their basic form. There are various methods for performing stemming, including lookup tables and suffix-stripping algorithms. Although stemming may not always yield desired conversions, it still aids in achieving text standardization.

```
# imports
import nltk# function for stemming
def get_stem(text):
    stemmer = nltk.porter.PorterStemmer()
    text = ' '.join([stemmer.stem(word) for word in text.split()])
    return text# call function
get_stem("we are eating and swimming ; we have been eating and
swimming ; he eats and swims ; he ate and swam ")
```

Output:

```
'we are eat and swim ; we have been eat and swim ; he eat and swim
; he ate and swam'
```

Fig 4.15 Stemming

4.3.8 Lemmatization

While both stemming and lemmatization aim to generate the root form of words, lemmatization is considered to be a more advanced technique compared to stemming. Stemming may not always produce actual words as the resulting stems, whereas lemmatization ensures proper word conversion by utilizing vocabulary and context. It focuses on removing inflectional endings and returning the base or dictionary form of a word, known as the lemma.However, it's important to note that lemmatization is generally slower than stemming, so the performance aspect should be considered when deciding between the two techniques. Stemming may be preferred in situations where speed is a priority, while lemmatization is typically chosen for more accurate and linguistically correct results.

```
# imports
import spacy
nlp = spacy.load('en',parse=True,tag=True, entity=True)# function
to remove special characters
def get_lem(text):
    text = nlp(text)
    text = ' '.join([word.lemma_ if word.lemma_ != '-PRON-' else
word.text for word in text])
    return text# call function
get_lem("we are eating and swimming ; we have been eating and
swimming ; he eats and swims ; he ate and swam ")
```

Output:

```
'we be eat and swim ; we have be eat and swim ; he eat and swim ;
he eat and swam'
```

Fig 4.16 Lammentization

4.4 REMOVING STOPWORDS

Stopwords are commonly used words that are often added to sentences to fulfill grammatical requirements but carry minimal to no semantic meaning. Examples of stopwords include words like "a," "is," "an," "the," "and," etc. In text processing tasks, it is beneficial to remove stopwords so that machine learning algorithms can focus on words that are more meaningful and contribute to the overall understanding of the text.

The NLTK library provides a list of commonly used stopwords that can be used for this purpose. However, it is worth noting that the stopwords list can be customized based on the specific context or requirements of the task. Additional stopwords can be added or existing ones can be removed to tailor the list to the particular application.

By removing stopwords, the resulting text data becomes more concise and relevant, allowing algorithms to focus on the words that carry more significant information and contribute to the desired analysis or processing task.

```
# imports
import nltk
from nltk.tokenize import ToktokTokenizer
tokenizer = ToktokTokenizer()
stopword_list = nltk.corpus.stopwords.words('english')
# custom: removing words from list
stopword_list.remove('not')# function to remove stopwords
def remove_stopwords(text):
    # convert sentence into token of words
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    # check in lowercase
    t = [token for token in tokens if token.lower() not in
stopword_list]
    text = ' '.join(t)
    return text# call function
remove_stopwords("i am myself you the stopwords list and this
article is not should removed")
```

Fig 4.17 Removing Stopwords

Output:

```
'stopwords list article not removed'
```

4.4.1 Removing extra whitespaces and tabs

Extra whitespaces and tabs do not add any information to text processing. Handling these should be fairly easy.

```
# imports
import re# function to remove special characters
def remove_extra_whitespace_tabs(text):
    #pattern = r'^\s+$|\s+$'
    pattern = r'^\s*|\s\s*'
    return re.sub(pattern, ' ', text).strip()# call function
remove_extra_whitespace_tabs('  This web line  has \t some extra
\t   tabs and whitespaces  ')
```

Output:

```
'This web line has some extra tabs and whitespaces'
```

Fig 4.18 Removing extra white spaces and tabs

4.4.2 Lowercase

Changing case to lower can be achieved by using lower function.

```
# function to remove special characters
def to_lowercase(text):
    return text.lower()# call function
to_lowercase('ConVert THIS string to LOWER cASe.')
```

Output:

```
'convert this string to lower case.'
```

Fig 4.19 Lowercase

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Fig 4.20 Stopwords

4.5 VECTOR REPRESENTATION OF SENTENCES

Word embeddings or word vectorization is a technique in natural language processing (NLP) that assigns numerical values to words or phrases. These numerical values represent the meaning or context of the words and are useful for tasks like predicting words, measuring word similarity, and analyzing text.

The process of converting words into numbers is called vectorization. Word embeddings have various applications, including finding similar words, classifying texts, grouping documents, extracting features for text analysis, and performing natural language processing tasks.

Once words are represented as vectors, we can use techniques like Euclidean distance or cosine similarity to determine their similarity. Cosine similarity is often preferred because it overcomes the limitations of counting common words or using Euclidean distance. Counting common words can lead to inaccurate results when documents discuss different topics, even if they have many words in common. Cosine similarity provides a more reliable measure of similarity between documents.

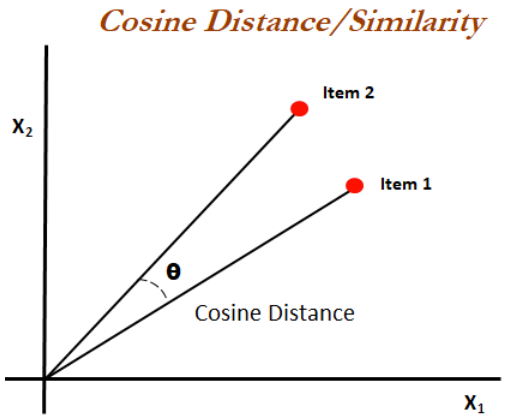


Fig 4.21 Cosine Similarity

Mathematically, cosine similarity calculates the cosine of the angle between two vectors (item1, item2) projected in an N-dimensional vector space. The cosine similarity has an advantage over Euclidean distance when it comes to predicting document similarity.

In cosine similarity, the smaller the angle between two vectors, the higher the similarity. To illustrate this, let's consider an example:

1. Julie loves John more than Linda loves John.
2. Jane loves John more than Julie loves John.

To determine the similarity between these two sentences using cosine similarity, we need to convert the sentences into vector representations. Let's assume we have already performed word embeddings and assigned vectors to the words in these sentences. For simplicity, let's represent each word with a single number:

1. Julie: [0.4], loves: [0.7], John: [0.9], more: [0.3], than: [0.2], Linda: [0.5]
2. Jane: [0.6], loves: [0.7], John: [0.9], more: [0.3], than: [0.2], Julie: [0.4]

Now, we calculate the cosine similarity between the vectors representing these sentences. The cosine similarity formula is:

cosine_similarity = (A dot B) / (||A|| * ||B||)

For the first sentence: (Julie loves John more than Linda loves John)
- Vector A = [0.4, 0.7, 0.9, 0.3, 0.2, 0.5]
- Vector B = [0.6, 0.7, 0.9, 0.3, 0.2, 0.4]

Using the cosine similarity formula, we find that the similarity between these two sentences is approximately 0.992.

This indicates a high degree of similarity between the two sentences, as the cosine similarity is close to 1. It suggests that the sentences share similar semantic meaning, despite some variations in the words used.

By employing cosine similarity, we can effectively measure the similarity between different sentences or documents, even when the exact wording differs.

4.6 SIMILARITY MATRIX

To compute the cosine similarity, you need the word count of the words in each document.

The `CountVectorizer` or the `TfidfVectorizer` from scikit learn lets us compute this. The output of this comes as a `sparse_matrix`.
On this,optionally converting it to a pandas dataframe to see the word frequencies in a tabular format.

```
# Scikit Learn
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Create the Document Term Matrix
count_vectorizer = CountVectorizer(stop_words='english')
count_vectorizer = CountVectorizer()
sparse_matrix = count_vectorizer.fit_transform(documents)

# OPTIONAL: Convert Sparse Matrix to Pandas Dataframe if you want
to see the word frequencies.
doc_term_matrix = sparse_matrix.todense()
df = pd.DataFrame(doc_term_matrix,
                  columns=count_vectorizer.get_feature_names(),
                  index=['doc_trump', 'doc_election', 'doc_putin'])
df
```

Fig 4.22 Similarity matrix

	after	as	became	by	career	claimed	do	earlier	election	elections	...	the	though	to	trump	vladimir
doc_trump	1	0	1	0	0	0	0	0	1	0	...	1	1	0	2	0
doc_election	0	0	0	1	0	1	1	0	2	0	...	2	0	1	1	0
doc_putin	0	1	1	0	1	0	0	1	0	1	...	1	0	0	0	1

3 rows × 48 columns

Fig 4.23 Doc-Term Matrix

Another improvement to the process would be to use the TfidfVectorizer() instead of the CountVectorizer(). The TfidfVectorizer not only considers the occurrence of words in documents but also downweights words that appear frequently across multiple documents. This helps in capturing the importance of words in a specific document.

After applying the TfidfVectorizer to convert the text into a document-term matrix, we can use the cosine_similarity() function to calculate the cosine similarity between the document vectors. This function accepts inputs in various formats, including pandas dataframes and sparse matrices.

By using the TfidfVectorizer and cosine_similarity together, we can obtain a more accurate measure of similarity between documents, taking into account both the frequency of words and their importance in the context of each document.

```
# Compute Cosine Similarity
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df, df))
#> [[ 1.          0.48927489  0.37139068]
#>  [ 0.48927489  1.          0.38829014]
#>  [ 0.37139068  0.38829014  1.        ]]
```

Fig 4.24 computation of cosine similarity

4.7. SUMMARY

Summary in text summarization refers to a concise representation of the main ideas or key points of a given text. It involves condensing a longer piece of text, such as an article or document, into a shorter version while retaining its essential meaning. The purpose of text summarization is to provide a brief overview or synopsis of the original text, enabling readers to grasp the main information without having to read the entire document.

Text summarization techniques analyze the content of the text, identify important sentences or phrases, and extract relevant information to create a summary. These techniques may consider factors like sentence relevance, importance of specific words or phrases, and overall coherence of the summary.

The goal of text summarization is to produce a concise and coherent summary that captures the essence of the original text. Summaries can be of various lengths, ranging from a few sentences to a paragraph, depending on the desired level of detail. Text summarization finds applications in various domains, including news articles, research papers, and online content, where it helps users quickly understand the main points of a text without having to read the entire document.

CHAPTER 5

EVALUATION AND TESTING

Output 1

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

Output 2

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

Output 3

		Unnamed: 0	id	title	publication	author	date	year	month	url	article
x	0	0	17283	House Republicans Fret About Winning Their Hea...	New York Times	Carl Hulse	2016-12-31	2016.0	12.0	NaN	WASHINGTON — Congressional Republicans have...
	1	1	17284	Rift Between Officers and Residents as Killing...	New York Times	Benjamin Mueller and Al Baker	2017-06-19	2017.0	6.0	NaN	After the bullet shells get counted, the blood...
	2	2	17285	Tyrus Wong, 'Bambi' Artist Thwarted by Racial ...	New York Times	Margalit Fox	2017-01-06	2017.0	1.0	NaN	When Walt Disney's "Bambi" opened in 1942, cri...
	3	3	17286	Among Deaths in 2016, a Heavy Toll in Pop Musi...	New York Times	William McDonald	2017-04-10	2017.0	4.0	NaN	Death may be the great equalizer, but it isn't...
	4	4	17287	Kim Jong-un Says North Korea Is Preparing to T...	New York Times	Choe Sang-Hun	2017-01-02	2017.0	1.0	NaN	SEOUL, South Korea — North Korea's leader, ...

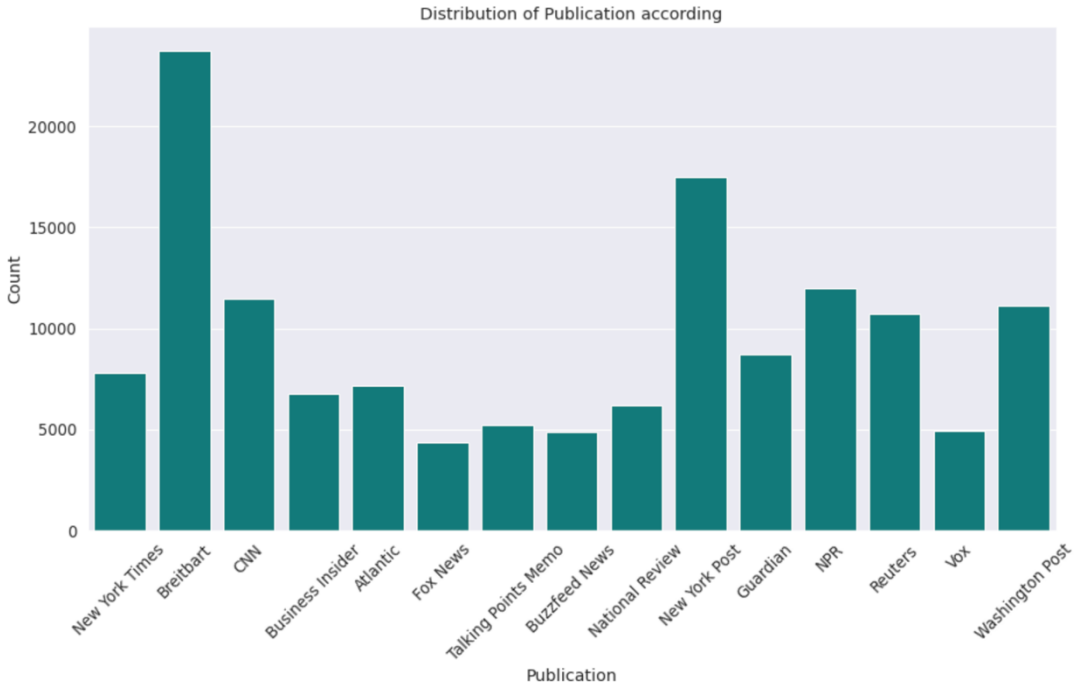
Output 4

		Unnamed: 0	id	title	publication	author	date	year	month	url	article
x	0	0	17283	House Republicans Fret About Winning Their Hea...	New York Times	Carl Hulse	2016-12-31	2016.0	12.0	NaN	WASHINGTON — Congressional Republicans have...
	1	1	17284	Rift Between Officers and Residents as Killing...	New York Times	Benjamin Mueller and Al Baker	2017-06-19	2017.0	6.0	NaN	After the bullet shells get counted, the blood...
	2	2	17285	Tyrus Wong, 'Bambi' Artist Thwarted by Racial ...	New York Times	Margalit Fox	2017-01-06	2017.0	1.0	NaN	When Walt Disney's "Bambi" opened in 1942, cri...
	3	3	17286	Among Deaths in 2016, a Heavy Toll in Pop Musi...	New York Times	William McDonald	2017-04-10	2017.0	4.0	NaN	Death may be the great equalizer, but it isn't...
	4	4	17287	Kim Jong-un Says North Korea Is Preparing to T...	New York Times	Choe Sang-Hun	2017-01-02	2017.0	1.0	NaN	SEOUL, South Korea — North Korea's leader, ...

Output 5

```
[Text(0.5, 0, 'Publication'),
 Text(0, 0.5, 'Count'),
 Text(0.5, 1.0, 'Distribution of Publication according')]
```

Output 6



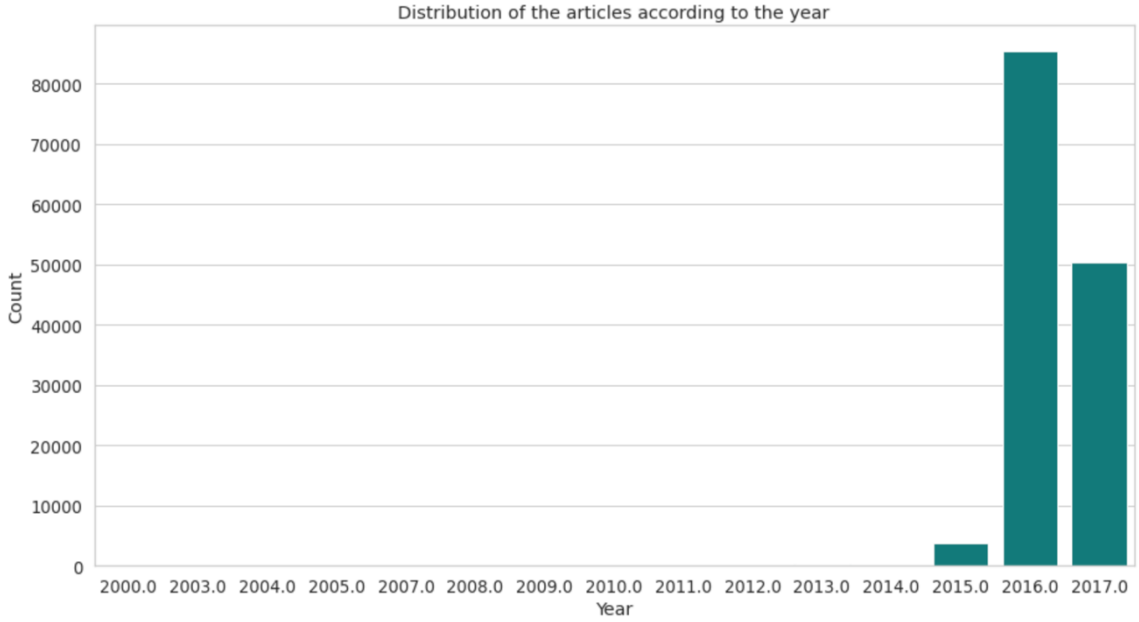
Output 7

```
2016.0    85405
2017.0    50404
2015.0     3705
2013.0      228
2014.0      125
2012.0       34
2011.0        8
2010.0        6
2008.0        3
2009.0        3
2004.0        2
2003.0        2
2005.0        2
2007.0        1
2000.0        1
Name: year, dtype: int64
```

Output 8

```
[Text(0.5, 0, 'Year'),
 Text(0, 0.5, 'Count'),
 Text(0.5, 1.0, 'Distribution of the articles according to the year')]
```

Output 9



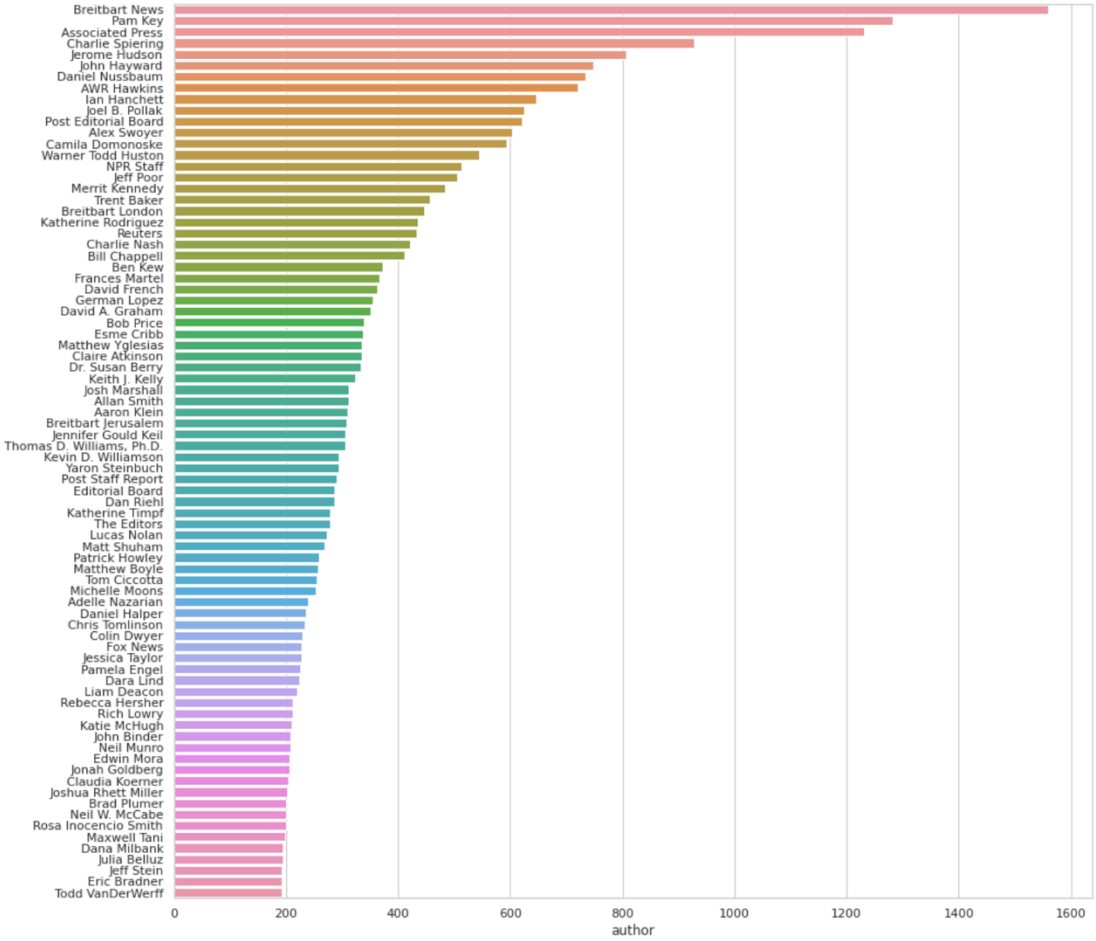
Output 10

```
Breitbart News                                  1559
Pam Key                                         1282
Associated Press                                1231
Charlie Spiering                                 928
Jerome Hudson                                    806
                                                 ...
Sylvie Bigar                                       1
Ludwig Burger, Greg Roumeliotis and Arno Schuetze  1
Kyle Pomerleau                                     1
Richard J. Light                                   1
F. Vincent Vernuccio                               1
Name: author, Length: 15647, dtype: int64
```

Output 11

```
[Text(0.5, 21.200000000000003, 'count'),
 Text(21.200000000000003, 0.5, 'author'),
 Text(0.5, 1.0, 'the most freq author')]
```

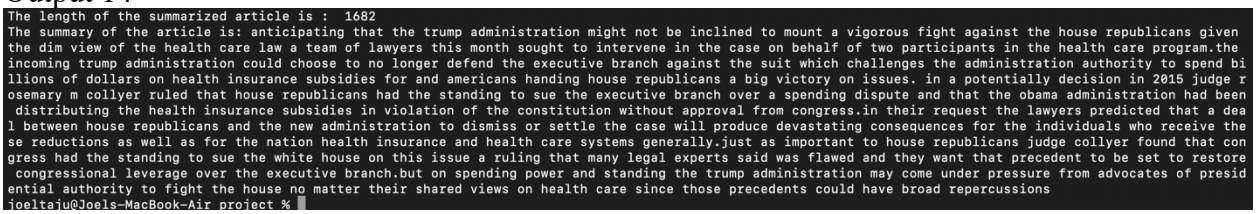
Output 12



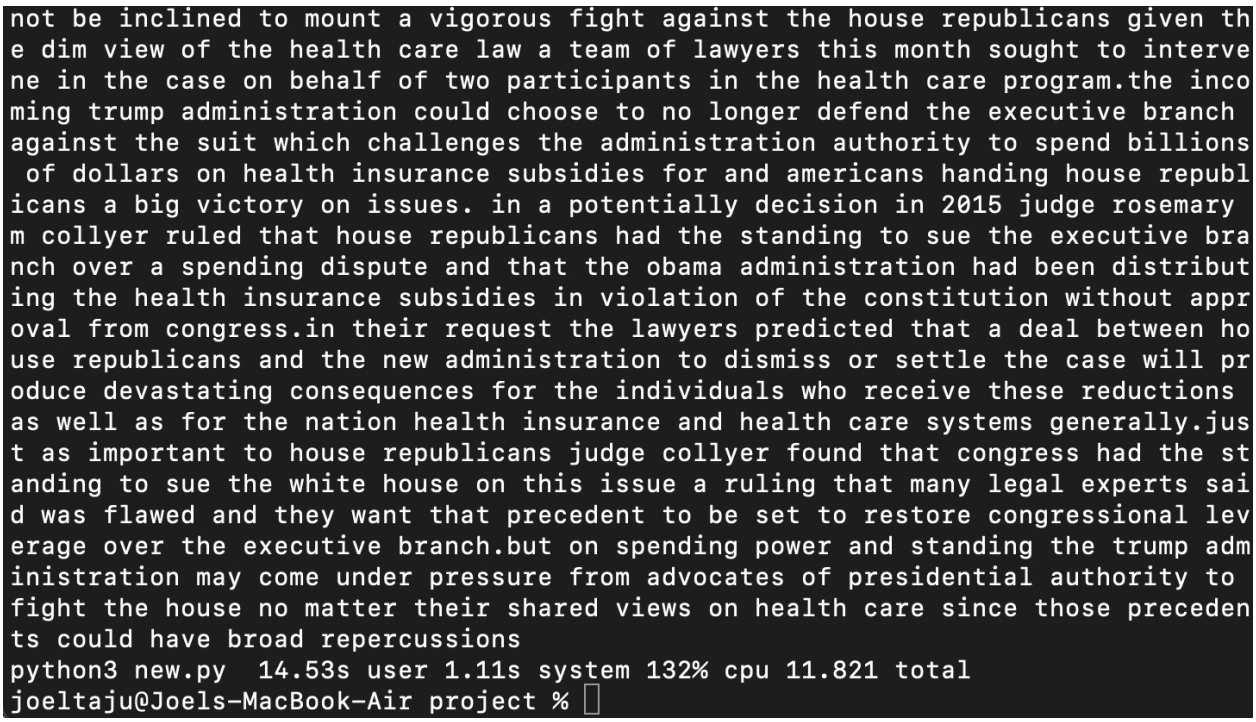
Output 13

WASHINGTON — Congressional Republicans have a new fear when it comes to their health care lawsuit against the Obama administration: They might win. The incoming Trump administration could choose to no longer defend the executive branch against the suit, which challenges the administration's authority to spend billions of dollars on health insurance subsidies for and Americans, handing House Republicans a big victory on issues. But a sudden loss of the disputed subsidies could conceivably cause the health care program to implode, leaving millions of people without access to health insurance before Republicans have prepared a replacement. That could lead to chaos in the insurance market and spur a political backlash just as Republicans gain full control of the government. To stave off that outcome, Republicans could find themselves in the awkward position of appropriating huge sums to temporarily prop up the Obama health care law, angering conservative voters who have been demanding an end to the law for years. In another twist, Donald J. Trump's administration, worried about preserving executive branch prerogatives, could choose to fight its Republican allies in the House on some central questions in the dispute. Eager to avoid an ugly political pileup, Republicans on Capitol Hill and the Trump transition team are gaming out how to handle the lawsuit, which, after the election, has been put in limbo until at least late February by the United States Court of Appeals for the District of Columbia Circuit. They are not yet ready to divulge their strategy. "Given that this pending litigation involves the Obama administration and Congress, it would be inappropriate to comment," said Phillip J. Blando, a spokesman for the Trump transition effort. "Upon taking office, the Trump administration will evaluate this case and all related aspects of the Affordable Care Act. " In a potentially decision in 2015, Judge Rosemary M. Collyer ruled that House Republicans had the standing to sue the executive branch over a spending dispute and that the Obama administration had been distributing the health insurance subsidies, in violation of the Constitution, without approval from Congress. The Justice Department, confident that Judge Collyer's decision would be reversed, quickly appealed, and the subsidies have remained in place during the appeal. In successfully seeking a temporary halt in the proceedings after Mr. Trump won, House Republicans last month told the court that they "and the ' s transition team currently are discussing potential options for resolution of this matter, to take effect after the 's inauguration on Jan. 20, 2017. " The suspension of the case, House lawyers said, will "provide the and his future administration time to consider whether to continue prosecuting or to otherwise resolve this appeal. " Republican leadership officials in the House acknowledge the possibility of "cascading effects" if the payments, which have totaled an estimated $13 billion, are suddenly stopped. Insurers that receive the subsidies in exchange for paying costs such as deductibles and for eligible consumers could race to drop coverage since they would be losing money. Over all, the loss of the subsidies could destabilize the entire program and cause a lack of confidence that leads other insurers to seek a quick exit as well. Anticipating that the Trump administration might not be inclined to mount a vigorous fight against the House Republicans given the 's dim view of the health care law, a team of lawyers this month sought to intervene in the case on behalf of two participants in the health care program. In their request, the lawyers predicted that a deal between House Republicans and the new administration to dismiss or settle the case "will produce devastating consequences for the individuals who receive these reductions, as well as for the nation's health insurance and health care systems generally. " No matter what happens, House Republicans say, they want to prevail on two overarching concepts: the congressional power of the purse, and the right of Congress to sue the executive branch if it violates the Constitution regarding that spending power. House Republicans contend that Congress never appropriated the money for the subsidies, as required by the Constitution. In the suit, which was initially championed by John A. Boehner, the House speaker at the time, and later in House committee reports, Republicans asserted that the administration, desperate for the funding, had required the Treasury Department to provide it despite widespread internal skepticism that the spending was proper. The White House said that the spending was a permanent part of the law passed in 2010, and that no annual appropriation was required — even though the administration initially sought one. Just as important to House Republicans, Judge Collyer found that Congress had the standing to sue the White House on this issue — a ruling that many legal experts said was flawed — and they want that precedent to be set to restore congressional leverage over the executive branch. But on spending power and standing, the Trump administration may come under pressure from advocates of presidential authority to fight the House no matter their shared views on health care, since those precedents could have broad repercussions. It is a complicated set of dynamics illustrating how a quick legal victory for the House in the Trump era might come with costs that Republicans never anticipated when they took on the Obama White House.

Output 14



Output 15



CHAPTER 6

CONCLUSION AND FUTURE WORK

- NLP has various applications and text summarization is one of the popular and great techniques.

- There are generally two approaches to achieve text summarization such as extractive summarization and abstractive summarization.

- It is an ancient challenge and the focus is shifting from extractive summarization to abstractive summarization.

- Abstractive summary technique generates relevant, exact, content full and less repetitive summary.

- Since the abstractive summarization technique focuses on generating summary which is nearer to the human intelligence & it is a challenging field.

REFERENCES

[1] Kam-Fai Wong, Mingli Wu – "Extractive Summarization Using Supervised and Semi-supervised Learning".

[2] Regina Barzilay, and Michael Elhadad – "Using lexical chains for text summarization".

[3] Md. Majharul Haque, Suraiya Pervin , and Zerina Begum – "Literature Review of Automatic Multiple Documents Text Summarization".

[4] S.A.Babar ,Pallavi D.Patil – "Improving Performance of Text Summarization".

[5] Nikhil S. Shirwandkar, Dr. Samidha Kulkarni – "Extractive Text Summarization using Deep Learning".

[6] Goularte,Fábio & Nassar, Silvia & Fileto, Renato & Saggion, Horacio – "A Text Summarization Method based on Fuzzy Rulesand applicable to Automated Assessment", Expert Systems with Applications. 115. 10.1016.

[7] Jo, Duke Taeho (2017) – "K nearest neighbor for text summarization using feature similarity", 1-5. 10.1109/ICCCCEE.2017.7866705.

[8] Anand, Deepa & Wagh, Rupali. (2019) – "Effective Deep Learning Approaches for Summarization of Legal Texts", Journal of King Saud University - Computer and Information Sciences.10.1016/j.jksuci.2019.11.015.

[9] P. Krishnaveni and S. R. Balasundaram - "Automatic text summarization by local scoring and ranking for improvingcoherence," 2017 International Conference on Computing Methodologies and Communication (ICCMC).

[10] J. Chen and H. Zhuge - "Extractive Text-Image Summarization Using Multi-Modal RNN," 2018 14th International Conference on Semantics, Knowledge and Grids (SKG), Guangzhou, China, 2018, pp. 245-248.doi: 10.1109/SKG.2018.00033.

[11] Valverde Tohalino, Jorge & Amancio, Diego. (2017) – "Extractive Multi-document Summarization Using Multilayer Networks", Physica A: Statistical Mechanics and its Applications 503.

[12] J. N. Madhuri and R. Ganesh Kumar - "Extractive Text Summarization Using Sentence Ranking," 2019 International Conference on Data Science and Communication (IconDSC), Bangalore, India, 2019, pp.1-3.doi: 10.1109/IconDSC.2019.8817040.

[13] N. S. Shirwandkar and S. Kulkarni - "Extractive Text Summarization Using Deep Learning," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA),Pune, India, 2018, pp. 1-5. doi: 10.1109/ICCUBEA.2018.8697465.

[14] Azar, Mahmood & Hamey, Len. (2016) - "Text Summarization Using Unsupervised Deep Learning", Expert Systems with Applications. 68. 10.1016/j.eswa.2016.10.017.

[15] Jain, Aditya & Bhatia, Divij & Thakur, Manish. (2017) – "Extractive Text Summarization Using Word Vector Embedding", 51-55. 10.1109/MLDS.2017.12.

[16] K.Selvani Deepthi, Dr.Radhika Y(2015) - "Extractive Text Summarization using Modified Weighing and Sentence SymmetricFeature Methods", in an International Journal of Modern Education and Computer Science, ISSN: 2075-0161 Volume 7 No-10 pp: 33-39.

[17] Supreetha. D , Rajeshwari. S. B , Jagadish. S Kalliman –" Abstractive Text Summarization Techniques".

[18] Rahul, SurabhiAdhikari and Monika - "NLP based machine learning approaches for text summarization",2020.

[19] B. Mutlu, E. A. Sezerand M. A. Akcayol - "Multi-document extractive text summarization: A comparative assessment on features," Knowledge-Based System. 104848, Vol. 183.

[20] M. Afsharizadeh, H. Ebrahimpour-Komlehand A. Bagheri - "Query-oriented text summarization using sentence extraction technique," Fourth Int. Conf. Web Res., pp. 128–132.

[21] M. Mauro, L. Canini, S. Benini, N. Adami, A. Signoroniand R. Leonardi - "A freeWeb API for single and multi-document summarization," ACM Int. Conf. Proceeding Ser., Vol. Part F1301.

[22] Atif Khan and Naomaisalim, - "A review on abstractive summarization methods".

[23] G. Silva, R. Ferreira, S. J. Simske, L. Rafael Lins, M. Rissand H. O. Cabral - "Automatic text document summarization based on machine learning," pp. 191–194.

[24] A. Jain, D. Bhatia and M. K. Thakur - "Extractive Text Summarization Using Word Vector Embedding," Proceedingsof International Conference on Machine Learning and Data Science, pp. 51– 55, Vol. 2018.

[25] Amjad Abu-Jbara and Dragomir Radev – "Coherent citation-based summarization of scientific papers", Human Language Technologies-Volume 1.Association for Computational Linguistics, 500–509.

[26] Rasim M Alguliev, Ramiz M Aliguliyev, and Nijat R Isazade – “Multiple documents summarization based on evolutionary optimization algorithm”, Expert Systems with Applications 40, 5 (2013), 1675–1689.

[27] Xiaojun Wan and Jianwu Yang – “Multidocument summarization using cluster-based link analysis”.

[28] Liang Zhou, Miruna Ticrea, and Eduard Hovy – “Multi-document biography Summarization”, arXiv preprint cs/0501078.

[29] Dingding Wang, Shenghuo Zhu, Tao Li, and Yihong Gong –“Multidocument summarization using sentence-based topic models”, In Proceedings of the ACL-IJCNLP Conference Short Papers. Association for ComputationalLinguistics, 297–300.

[30] Rada Mihalcea and Paul Tarau – “A language independent algorithm for single and multiple document summarization”.

ANNEXURE

```
# -*- coding: utf-8 -*-
import numpy as np
import pandas as pd
import warnings
import re
import nltk
from nltk import word_tokenize
from nltk.tokenize import sent_tokenize
from textblob import TextBlob
import string
from string import punctuation
from nltk.corpus import stopwords
from statistics import mean
from heapq import nlargest
from wordcloud import WordCloud

stop_words = set(stopwords.words('english'))
punctuation = punctuation + '\n' + '—' + '“' + ',' + '”' + '‘' + '–' + '’'
warnings.filterwarnings('ignore')

# Importing the dataset
df_1 = pd.read_csv("articles1.csv")
df_2 = pd.read_csv("articles2.csv")
df_3 = pd.read_csv("articles3.csv")

# Checking if the columns are same or not
(df_1.columns == df_2.columns)

# Checking if the columns are same or not
(df_2.columns == df_3.columns)

# Making one Dataframe by appending all of them for the further process
d = [df_1, df_2, df_3]
df = pd.concat(d, keys=['x', 'y', 'z'])
df.rename(columns={'content': 'article'}, inplace=True)
(df.head())

# Shape of the dataset
("The shape of the dataset : ", df.shape)

# Dropping the unnecessary columns
df.drop(columns=['Unnamed: 0'], inplace=True)
df.head()

# Countplot shows the distribution of Publication

import seaborn as sns
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [15, 8]
sns.set(font_scale=1.2, style="darkgrid")
sns_year = sns.countplot(data=df, x="publication", color="darkcyan")
plt.xticks(rotation=45)
sns_year.set(xlabel="Publication", ylabel="Count", title="Distribution of Publication")
# plt.show()


# Replacing the unnecessary row value of year with it's actual values
df['year'] = df['year'].replace(
    "https://www.washingtonpost.com/outlook/tale-of-a-woman-who-died-and-a-woman-who-killed-in-the-northern-ireland-conflict/2019/03/08
    2019)

# Years
(df['year'].value_counts())

# Countplot shows the distribution of the articles according to the year


plt.rcParams['figure.figsize'] = [15, 8]
sns.set(font_scale=1.2, style='whitegrid')
```

```
sns_year = sns.countplot(data=df, x="year", color='darkcyan')
sns_year.set(xlabel="Year", ylabel="Count", title="Distribution of the articles according to the year")
# plt.show()


# Authors
df['author'].value_counts()

import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams['figure.figsize'] = [15, 15]
sns.set(font_scale=1, style='whitegrid')

df_author = df.author.value_counts().head(80)

sns.barplot(x=df_author.values, y=df_author.index)
plt.xlabel("Count",fontsize=12)
plt.ylabel("Author")
plt.title("The Most Frequent Authors")
sns.despine(left=True)
plt.yticks(fontsize=8)

plt.grid(True,axis='both')  # Enable grid lines

sns.set_style('ticks')  # Set the style to 'ticks'
plt.xticks(rotation=0, ha='center')  # Adjust rotation and alignment of x-axis labels

plt.tight_layout()
#plt.show()


# Changing the value "The Associated Press" to "Associated Press"
df['author'] = df['author'].replace("The Associated Press", "Associated Press")


contractions_dict = {
"ain't": "am not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"doesn’t": "does not",
"don't": "do not",
"don’t": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he had",
"he'd've": "he would have",
"he'll": "he will",
"he'll've": "he will have",
"he's": "he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how is",
"i'd": "i would",
"i'd've": "i would have",
"i'll": "i will",
"i'll've": "i will have",
"i'm": "i am",
"i've": "i have",
"isn't": "is not",
"it'd": "it would",
```

```
    "it'd": "it would",
    "it'd've": "it would have",
    "it'll": "it will",
    "it'll've": "it will have",
    "it's": "it is",
    "let's": "let us",
    "ma'am": "madam",
    "mayn't": "may not",
    "might've": "might have",
    "mightn't": "might not",
    "mightn't've": "might not have",
    "must've": "must have",
    "mustn't": "must not",
    "mustn't've": "must not have",
    "needn't": "need not",
    "needn't've": "need not have",
    "o'clock": "of the clock",
    "oughtn't": "ought not",
    "oughtn't've": "ought not have",
    "shan't": "shall not",
    "sha'n't": "shall not",
    "shan't've": "shall not have",
    "she'd": "she would",
    "she'd've": "she would have",
    "she'll": "she will",
    "she'll've": "she will have",
    "she's": "she is",
    "should've": "should have",
    "shouldn't": "should not",
    "shouldn't've": "should not have",
    "so've": "so have",
    "so's": "so is",
    "that'd": "that would",
    "that'd've": "that would have",
    "that's": "that is",
    "there'd": "there would",
    "there'd've": "there would have",
    "there's": "there is",
    "they'd": "they would",
    "they'd've": "they would have",
    "they'll": "they will",
    "they'll've": "they will have",
    "they're": "they are",
    "they've": "they have",
    "to've": "to have",
    "wasn't": "was not",
    "we'd": "we would",
    "we'd've": "we would have",
    "we'll": "we will",
    "we'll've": "we will have",
    "we're": "we are",
    "we've": "we have",
    "weren't": "were not",
    "what'll": "what will",
    "what'll've": "what will have",
    "what're": "what are",
    "what's": "what is",
    "what've": "what have",
    "when's": "when is",
    "when've": "when have",
    "where'd": "where did",
    "where's": "where is",
    "where've": "where have",
    "who'll": "who will",
    "who'll've": "who will have",
    "who's": "who is",
    "who've": "who have",
    "why's": "why is",
    "why've": "why have",
```

```
  "will've": "will have",
  "won't": "will not",
  "won't've": "will not have",
  "would've": "would have",
  "wouldn't": "would not",
  "wouldn't've": "would not have",
  "y'all": "you all",
  "y’all": "you all",
  "y'all'd": "you all would",
  "y'all'd've": "you all would have",
  "y'all're": "you all are",
  "y'all've": "you all have",
  "you'd": "you would",
  "you'd've": "you would have",
  "you'll": "you will",
  "you'll've": "you will have",
  "you're": "you are",
  "you've": "you have",
  "ain't": "am not",
  "aren't": "are not",
  "can't": "cannot",
  "can't've": "cannot have",
  "'cause": "because",
  "could've": "could have",
  "couldn't": "could not",
  "couldn't've": "could not have",
  "didn't": "did not",
  "doesn't": "does not",
  "don't": "do not",
  "don't": "do not",
  "hadn't": "had not",
  "hadn't've": "had not have",
  "hasn't": "has not",
  "haven't": "have not",
  "he'd": "he had",
  "he'd've": "he would have",
  "he'll": "he will",
  "he'll've": "he will have",
  "he's": "he is",
  "how'd": "how did",
  "how'd'y": "how do you",
  "how'll": "how will",
  "how's": "how is",
  "i'd": "i would",
  "i'd've": "i would have",
  "i'll": "i will",
  "i'll've": "i will have",
  "i'm": "i am",
  "i've": "i have",
  "isn't": "is not",
  "it'd": "it would",
  "it'd've": "it would have",
  "it'll": "it will",
  "it'll've": "it will have",
  "it's": "it is",
  "let's": "let us",
  "ma'am": "madam",
  "mayn't": "may not",
  "might've": "might have",
  "mightn't": "might not",
  "mightn't've": "might not have",
  "must've": "must have",
  "mustn't": "must not",
  "mustn't've": "must not have",
  "needn't": "need not",
  "needn't've": "need not have",
  "o'clock": "of the clock",
  "oughtn't": "ought not",
  "oughtn't've": "ought not have",
  "shan't": "shall not",
  "sha'n't": "shall not",
```

```
281   "shan't've": "shall not have",
282   "she'd": "she would",
283   "she'd've": "she would have",
284   "she'll": "she will",
285   "she'll've": "she will have",
286   "she's": "she is",
287   "should've": "should have",
288   "shouldn't": "should not",
289   "shouldn't've": "should not have",
290   "so've": "so have",
291   "so's": "so is",
292   "that'd": "that would",
293   "that'd've": "that would have",
294   "that's": "that is",
295   "there'd": "there would",
296   "there'd've": "there would have",
297   "there's": "there is",
298   "they'd": "they would",
299   "they'd've": "they would have",
300   "they'll": "they will",
301   "they'll've": "they will have",
302   "they're": "they are",
303   "they've": "they have",
304   "to've": "to have",
305   "wasn't": "was not",
306   "we'd": "we would",
307   "we'd've": "we would have",
308   "we'll": "we will",
309   "we'll've": "we will have",
310   "we're": "we are",
311   "we've": "we have",
312   "weren't": "were not",
313   "what'll": "what will",
314   "what'll've": "what will have",
315   "what're": "what are",
316   "what's": "what is",
317   "what've": "what have",
318   "when's": "when is",
319   "when've": "when have",
320   "where'd": "where did",
321   "where's": "where is",
322   "where've": "where have",
323   "who'll": "who will",
324   "who'll've": "who will have",
325   "who's": "who is",
326   "who've": "who have",
327   "why's": "why is",
328   "why've": "why have",
329   "will've": "will have",
330   "won't": "will not",
331   "won't've": "will not have",
332   "would've": "would have",
333   "wouldn't": "would not",
334   "wouldn't've": "would not have",
335   "y'all": "you all",
336   "y'all": "you all",
337   "y'all'd": "you all would",
338   "y'all'd've": "you all would have",
339   "y'all're": "you all are",
340   "y'all've": "you all have",
341   "you'd": "you would",
342   "you'd've": "you would have",
343   "you'll": "you will",
344   "you'll've": "you will have",
345   "you're": "you are",
346   "you're": "you are",
347   "you've": "you have",
348   }
349   import re
350   contractions_re = re.compile('(%s)' % '|'.join(contractions_dict.keys()))
351   # Function to clean the html from the article
```

```
# Function to clean the html from the article
def cleanhtml(raw_html):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', raw_html)
    return cleantext

# Function expand the contractions if there's any
def expand_contractions(s, contractions_dict=contractions_dict):
    def replace(match):
        return contractions_dict[match.group(0)]
    return contractions_re.sub(replace, s)

# Function to preprocess the articles
def preprocessing(article):
    global article_sent

    # Converting to lowercase
    article = article.str.lower()

    # Removing the HTML
    article = article.apply(lambda x: cleanhtml(x))

    # Removing the email ids
    article = article.apply(lambda x: re.sub('\S+@\S+','', x))

    # Removing The URLS
    article = article.apply(lambda x: re.sub("((http\://|https\://|ftp\://)|(www.))+(([a-zA-Z0-9\.-]+\.[a-zA-Z]{2,4})|([0-9]{1,3}\.[0-9

    # Removing the '\xa0'
    article = article.apply(lambda x: x.replace("\xa0", " "))

    # Removing the contractions
    article = article.apply(lambda x: expand_contractions(x))

    # Stripping the possessives
    article = article.apply(lambda x: x.replace("'s", ''))
    article = article.apply(lambda x: x.replace('’s', ''))
    article = article.apply(lambda x: x.replace("\'s", ''))
    article = article.apply(lambda x: x.replace("\’s", ''))

    # Removing the Trailing and leading whitespace and double spaces
    article = article.apply(lambda x: re.sub(' +', ' ',x))

    # Copying the article for the sentence tokenization
    article_sent = article.copy()

    # Removing punctuations from the article
    article = article.apply(lambda x: ''.join(word for word in x if word not in punctuation))

    # Removing the Trailing and leading whitespace and double spaces again as removing punctuation might
    # Lead to a white space
    article = article.apply(lambda x: re.sub(' +', ' ',x))

    # Removing the Stopwords
    article = article.apply(lambda x: ' '.join(word for word in x.split() if word not in stop_words))

    return article

# Function to normalize the word frequency which is used in the function word_frequency
def normalize(li_word):
    global normalized_freq
    normalized_freq = []
    for dictionary in li_word:
        max_frequency = max(dictionary.values())
        for word in dictionary.keys():
            dictionary[word] = dictionary[word]/max_frequency
        normalized_freq.append(dictionary)
    return normalized_freq

```

```
# Function to calculate the word frequency
def word_frequency(article_word):
    word_frequency = {}
    li_word = []
    for sentence in article_word:
        for word in word_tokenize(sentence):
            if word not in word_frequency.keys():
                word_frequency[word] = 1
            else:
                word_frequency[word] += 1
        li_word.append(word_frequency)
        word_frequency = {}
    normalize(li_word)
    return normalized_freq

# Function to Score the sentence which is called in the function sent_token
def sentence_score(li):
    global sentence_score_list
    sentence_score = {}
    sentence_score_list = []
    for list_, dictionary in zip(li, normalized_freq):
        for sent in list_:
            for word in word_tokenize(sent):
                if word in dictionary.keys():
                    if sent not in sentence_score.keys():
                        sentence_score[sent] = dictionary[word]
                    else:
                        sentence_score[sent] += dictionary[word]
        sentence_score_list.append(sentence_score)
        sentence_score = {}
    return sentence_score_list

# Function to tokenize the sentence
def sent_token(article_sent):
    sentence_list = []
    sent_token = []
    for sent in article_sent:
        token = sent_tokenize(sent)
        for sentence in token:
            token_2 = ''.join(word for word in sentence if word not in punctuation)
            token_2 = re.sub(' +', ' ',token_2)
            sent_token.append(token_2)
        sentence_list.append(sent_token)
        sent_token = []
    sentence_score(sentence_list)
    return sentence_score_list

# Function which generates the summary of the articles (This uses the 20% of the sentences with the highest score)
def summary(sentence_score_0w0):
    summary_list = []
    for summ in sentence_score_0w0:
        select_length = int(len(summ)*0.25)
        summary_ = nlargest(select_length, summ, key = summ.get)
        summary_list.append(".".join(summary_))
    return summary_list


# Functions to change the article string (if passed) to change it to generate a pandas series
def make_series(art):
    global dataframe
    data_dict = {'article' : [art]}
    dataframe = pd.DataFrame(data_dict)['article']
    return dataframe

# Function which is to be called to generate the summary which in further calls other functions alltogether
def article_summarize(artefact):

    if type(artefact) != pd.Series:
        artefact = make_series(artefact)

    df = preprocessing(artefact)
```

```
491
492         word_normalization = word_frequency(df)
493
494         sentence_score_OwO = sent_token(article_sent)
495
496         summarized_article = summary(sentence_score_OwO)
497
498         return summarized_article
499
500
501     # Generating the Word Cloud of the article using the preprocessing and make_series function mentioned below
502     def word_cloud(art):
503         art_ = make_series(art)
504         OwO = preprocessing(art_)
505         wordcloud_ = WordCloud(height = 500, width = 1000, background_color = 'white').generate(art)
506         plt.figure(figsize=(15, 10))
507         plt.imshow(wordcloud_, interpolation='bilinear')
508         plt.axis('off');
509     # Generating the summaries for the first 100 articles
510     summaries = article_summarize(df['article'][0:100])
511
512     print ("The Actual length of the article is : ", len(df['article'][0]))
513     print(df['article'][0])
514
515
516     print ("The length of the summarized article is : ", len(summaries[0]))
517     summaries[0]
518     print("The summary of the article is:", summaries[0])
519
520
521
```