WEB APP FOR EXTRACTIVE TEXT SUMMARIZATION FOR NEWS ARTICLES

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

In

Computer Science and Engineering/Information Technology

By

(Bhaskar Sen, 171247)

Under the supervision of

(Dr. Amit Kumar)

To

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234.

Certificate

Candidate's Declaration

I hereby declare that the work presented in this report entitled "**Web App Extractive Text Summarization of News Articles**" in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2020 to December 2020 under the supervision of (**Dr Amit Kumar**) (Assistant Proffesor(Senior Grade), Departmentt of CSE).
The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)

Bhaskar Sen, 171247

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)
Supervisor Name: Dr Amit Kumar
Designation: Assistant Proffesor(Senior Grade)
Department name: Computer Science and Engineering
Dated:

ACKNOWLEDGEMENT

I would like to take the opportunity to thank and express my deep sense of gratitude to my mentor and project guide Dr Pradeep Kumar Gupta for his immense support and valuable guidance without which it would not have been possible to reach at this stage of our final year project. I am also obliged to all my faculty members for their valuable support in their respective fields which helped me in reaching at this stage of my project.My thanks and appreciations also go to my colleagues who have helped me out with their abilities in developing the project.

Date: 16 June 2021

Bhaskar Sen

Table of Contents

List Of Abbreviations

AggSim----------Aggregate-Similarity

BushyP----------Bushy

PathCueP----------Cue-Phrase

LexicalS----------Lexical-Similarity

NumData----------Numerical-Data

PropNoun----------Proper-Noun

SCentral----------Sentence Centrality

SenLength----------Sentence Length

SPosition----------Sentence-Position

ResTitle----------Resemblance-Title

TextRankS----------TextRank-Score

TFIDF----------TF/IDF

UpCase----------Upper-Case

WCOcurrency----------Word Co-Occurrence

WF----------Word Frequency

List of Figures

List of Graphs

List of Tables

Abstract

Text summarization is the process of creating a shorter version of the text with only vital information and thus, helps the user to understand the text in a shorter amount of time. The main advantage of text summarization lies in the fact that it reduces user's time in searching the important details in the document.

There are two main approaches to summarizing text documents –

Extractive Method: It involves selecting phrases and sentences from the original text and including it in the final summary.

Example:

> **Original Text** : Python is a high-level, interpreted, interactive and object-oriented scripting language.Python is a great language for the beginner-level programmers.
>
> **Extractive Summary** : Python is a high-level scripting language is great language for beginner-level programmers.

Abstractive Method: The Abstractive method involves generating entirely new phrases and sentences to capture the meaning of source document.

Example:

> **Original Text** : Python is a high-level, interpreted, interactive and object-oriented scripting language.Python is a great language for the beginner-level programmers
>
> **Abstractive Summary** : Python is interpreted and interactive language and it is easy to learn.

Abstract

Text summarization is the process of creating a shorter version of the text with only vital information and thus, helps the user to understand the text in a shorter amount of time.The main advantage of text summarization lies in the fact that it reduces user's time in searching the important details in the document.There are two main approaches to summarizing text documents –

Extractive Method: It involves selecting phrases and sentences from the original text and including it in the final summary. Example:

Original Text : Python is a high-level, interpreted, interactive and object-oriented

scripting language.Python is a great language for the beginner-level programmers.

Extractive Summary : Python is a high-level scripting language is great language for beginner-level programmers.

Abstractive Method: The Abstractive method involves generating entirely new phrases and sentences to capture the meaning of source document. Example:

Original Text : Python is a high-level, interpreted, interactive and object-oriented scripting language.Python is a great language for the beginner-level programmers

Abstractive Summary : Python is interpreted and interactive language and it is easy to learn.

Chapter 1 Introduction

1.1 Introduction:

The quantity of information on the internet is massively increasing and gigantic volume of data with numerous compositions accessible openly online become more widespread. It is challenging nowadays for a user to extract the information efficiently and smoothly. As one of the methods to tackle this challenge, text summarization process diminishes the redundant information and retrieves the useful and relevant information from a text document or any news article to form a compressed and shorter version which is easy to understand and time-saving while reflecting the main idea of the discussed topic within the document. The approaches of automatic text summarization earn a keen interest within the Text Mining and NLP (Natural Language Processing) communities because it is a laborious job to manually summarize a text document. Mainly there are two types of text summarization, namely **extractive** based and **abstractive** based. We have used .**"Text-Rank"** algorithm which is derived originally from the "Page-Rank" algorithm, to generate an extractive summary of the text. We have used text from different news articles as the input.

1.2 Problem Statement:

We need to summarize the text of any given article which is used as input to our model. The text should be in *English* language only. The summary generated is an extract of the actual text and uses the Text-Rank algorithm.

1.3 Objective(s):

1. Import all necessary libraries.
2. Generate clean sentences.
3. Generate Similarity matrix
4. Generate Summary Method

1.4 Methodology:

We have used the TextRank algorithm for our model. TextRank is an extractive and unsupervised text summarization technique. Let's take a look at the flow of the TextRank algorithm that we will be following:

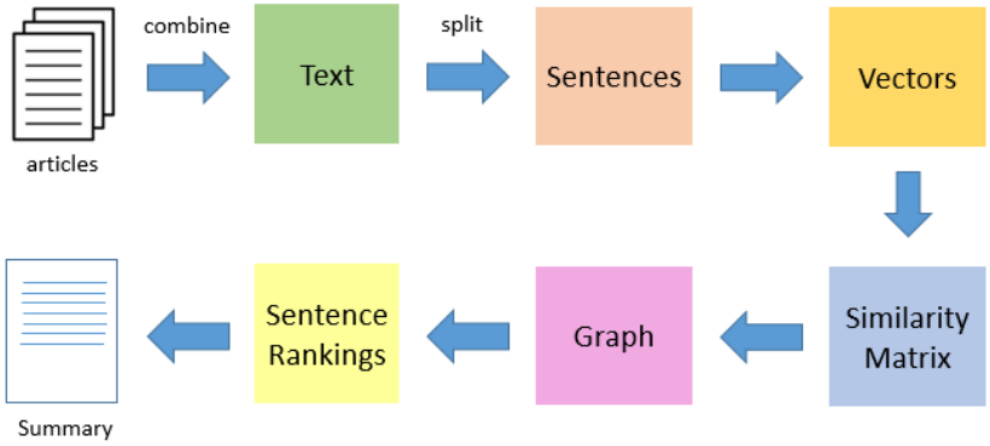


Fig 1

1. The first step would be to concatenate all the text contained in the articles
2. Then split the text into individual sentences
3. In the next step, we will find vector representation (word embeddings) for each and every sentence
4. Similarities between sentence vectors are then calculated and stored in a matrix
5. The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation
6. Finally, a certain number of top-ranked sentences form the final summary

Chapter 2 Literature Survey

[1] Rada Mihalcea, Department of Computer Science and Engineering University of North Texas. In her paper, "Language Independent Extractive Summarization" quotes:

> TextRank is a kind of unsupervised class of algorithms that is used for extractive summarization. It uses method of iterative network-ranking methods to encode the rigidly connected nature of texts. Another vital nature is that it dependent on any prior language-specific knowledge or any training data, and hence this makes it highly portable across various languages.

Living in an age of large data intensive articles and images, on the Web and elsewhere, methods for efficient summarizing the text database is essential to broaden the reach of such intensive information. Methods for extractive summarization are typically based on sentence scoring, and try to weigh the sentences that are most important for the interpretation of a given article. The more efficient and better methods of extractive summarization use supervised algorithms that learn what makes a better summary from training set of summaries built om a relatively large number of training documents,However, the cost of the of such supervised algorithms lies in their limitaion adapt to other languages and domains,. TextRank is addresses this problem, by employing an extractive summarization method that does not require any training data or any language oriented deductions. TextRank can be used for summarizing texts in various languages without making any changes in the methods and without any additional knowledge.

Extractive Summarization:

Ranking processes, such as *Kleinberg's HITS algorithm* or *Google's PageRank are being generally* used in link articulation, i.e, a network-based ranking algorithm , considering all the present data constantly ranked from the entire network, rather rather than always using a local node-specific point. Ranking model is for *voting* or *recommendation*. When 1node points to another one, it is adding a vote for that other node. Greater number of votes that are cast for a node, more is the rank(importance) of the node.

These network ranking algorithms are based on a random walk model, wherein a crawler takes shifts over the network, such walk is known as a Markov process – i.e, the decision on what relation to choose is completely based upon the node where the crawler is currently at. Mostly, This model rests to a constant distribution of probabilities associated with nodes in the network, i.e the probability of locating the crawler at a certain node in the network. Based on the theorem for Markov chains, the are guaranteed to rest if the network is not any further reducible and does not show any periodic properties. The first is find any network that is a non-bipartite network, while the 2nd holds for any well connected network. Both these are achieved in the networks for the extractive summarization application implemented in *TextRank*.

Given a directed network $S = (V, E)$ with In (Vi) denoting the set of nodes that point towards a node Vi, and $Out(Vi)$ denoting the set of nodes that node Vi points outwards ,the textrank score associated with each node defined as:

$$PR(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{PR(V_j)}{|Out(V_j)|} \qquad (1)$$

where d is a damping factor that is set between 0 and 1 ,includes random probing into the random walking model. Arbitrary values are assigned to each node in the network, the method iterates until reaches below a given threshold. Afterwards a score is given to each node, which is weightage of that node within the network. The final scores are not changed by the choice of starting node, only the number of iterations may be different.

When the networks is built using natural language texts, it is always profitable to add into the network model the associated inter-weights of the relation between two nodes V_I and V_j. such as a weight w_{ij} in the relation in between nodes . The scoring process is hence :

$$PR^W(V_i) = (1-d)+d* \sum_{V_j \in In(V_i)} w_{ji} \frac{PR^W(V_j)}{\sum_{V_k \in Out(V_j)} w_{kj}} \quad (2)$$

For case of a singular text extractive summarization, the motive is to rank the sentences in a given article wrt. Their contribution for the overall meaning of the text. A network is therefore made by considering node for each sentence in the article, and relations between nodes are formed using sentence contextual-relations. These relations are formed using a similarity relation, “similarity” can be called as a function of contextual overlay. Relation between two sentences can considered as a voting process: a sentence delivers some meaning to reader, gives the reader a vote for other sentences in the text that have the same meaning, and hence similarity can be found between those two sentences.

The similarity inbetween two sentences can be found by looking at the number of common lexical tokens between grammatical representations of sentences, it can be also be passed through other functions that remove stopwords, count only words of a certain category, etc. Also, to not to include long sentences, we can use a normalization, and divide the meaningful sentences with their span. The resulting network is highly

bonded, with a mass associated with each relation, indicating the strength of the relation between various sentence pairs in the text.

TextRank Algorithm:

The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.

The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.

The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.

$$Sim(S_i, S_j) = \frac{|\{w_k : w_k \in S_i \wedge w_k \in S_j\}|}{log(|S_i|) + log(|S_j|)}$$

[3] Chirantana Mallick, Ajit Kumar Das, Madhurima Dutta, Asit Kumar Das and Apurba Sarkar in their paper,"Graph-Based Text Summarization Using Modified TextRank", quote:

Since, the amount of data that is present on the internet is rapidly increasing day by day, we need some good algorithms to find and extract the useful piece of information from this huge amount data.

Data can be anything from photos, videos. Music. Spreadhsheets. Etc. This paper primarily focuses on an Algorithm that uses '**Graphs'** to summarize '**Text**' form of data.
A graph is generated between the sentences using,

Node: The sentence itself.

Edge: The similarity between two sentences.

Text summarization, reatins the origanl essence of the text while ingoring the least significant words in the original text.

They're two major summarization techniques. Known as:

Abstractive

Extractie.

Abstractive as the name suggests, is summarization in which we add our own grammar in the summarized text. Summarized text may or may not conatin the orignal phrases from the input text, wherears Extractive summarization uses only the original components in the text.

Abstractive is always preffered because of the face that it creates more huma-readable summaries.

Abstractive text summarization also retains the emotions of the author.

The other form of classifications is:

Indicative.

Informative.

Indicative Summary uses the main topics of the whole text as the summarized output where as Informative Summary emphasis on the meaning, and paraphrase of the text.
Informative summaries are generic in nautre. Query-oriented summaries generate a query set that reflects user's interest.

Intermediate Representation:

Even for the simplest summarizer, intermediate representation is necessary.
It involves the TF-IDF appraoch and topic word apporach.

TF stands for Term Frequency.

IDF stands for Inverse Document Freqency.

The topic representation of topic-word approaches consists of a simple list of words and their corresponding weights where higher weighted words are considered as indicative topic words. There are garph based models in which the whole set of setences are represented by a graph. Eg: LexRank, In indicator representation approaches, the input text document is represented as a list of indicators of such as sentence length, presence of certain phrases pertaining to the theme of the document, location in the document.

Score Sentences:

After the sentences are converted, they're given a special score usng the algorithms priority.

Algo such Text-Rank are used in this case.

Selecting Summary Sentences:

A matrix is thus created in which all the sentences are used in order of the score.

Then top N sentences are used to generate the summary.

[4]Rahim Khan, Yurong Qian, Sajid Naeem School of Software, Xinjiang University, Urumqi 830008, China, in their paper , "Extractive based Text Summarization Using K-Means and TF-IDF" quote:

Since, the amount of data that is present on the internet is rapidly increasing day by day, we need some good algorithms to find and extract the useful piece of information from this huge amount data.

It is moving these days for a client to extricate the data productively and easily. As one of the strategies to handle this test, text outline measure decreases the repetitive data and recovers the helpful and applicable data from a book report to shape a packed and more limited variant which is straightforward and efficient while mirroring the principle thought of the talked about subject inside the record. The approaches of automatic text summarization earn a keen interest within the Text Mining and NLP (Natural Language Processing) communities because it is a laborious job to manually summarize a text document.

The huge amount of data is rapidly increasing because of the availbility of cheap internet and free copy-right free text, audio and video files.

HLT(Human Language Technology) enables users/humans to interct with the machine. But since the size of data is rapidly incerasing, It becomes very difficult to cope with this huge amount of data.

They're major two algorithms which deal with summariazation. One of them being with the use of TF-IDF and K-means clustering.
The approach has been discussed in the paper with great detail.

[5] "What is a simple but detailed explanation of Textrank?" answered by, Luis Argerich, Professor of Data Science at the University of Buenos Aires. (UBA).

A non-numerical way to deal with TextRank (or construct your own content summarizer without networks)

Disclaimer: I will give an extremely basic yet exact portrayal of the calculation in light of the fact that the client mentioned it.

In light of that, here we go:

The fundamental thought of TextRank is to give a score to each sentence in a content, at that point you can take the top-n sentences and sort them as they show up in the content to assemble a programmed outline.

In the event that you apply it to the rundown of Star Wars 7, the power stirs and confine it to only one sentence it makes:

"Kylo Ren faces Finn and Rey in the cold timberland."

Which is quite great. So how about we perceive how to do that.

The initial step is obviously to separate all the sentences from the content. This can be as basic as parting the content at "." or newlines or more intricate in the event that you need to refine precisely what a sentence is. Parsers are rarely ended, they are simply surrendered. When you have all the sentences in the content you need to fabricate a diagram, in the chart each sentence is a hub and you put joins from each sentence to all others or to the k-most comparative sentences weighted by likeness. This is the place where I will allow you to utilize your imagination since you need to characterize a closeness score between two sentences, it tends to be a such thing as number of words in like manner over the all out number of words or a perplexing equation, this is the place where you will invest the greater part of your energy to "tune" textrank to function admirably for whatever you are attempting to sum up. Be careful: Different similitude scoring capacities will drastically change the consequence of TextRank.

When you have the weighted chart you need to run PageRank on it, since you are disclosing to us you are confounded by the mathematical I will give a straightforward estimation to PageRank.

Execute irregular strolls over the chart, in every arbitrary walk you start from an arbitrary hub, from that hub dependent on the loads you select an arbitrary neighbor, etc. For instance on the off chance that you are in hub "A" (recall every hub speaks to a sentence) and your neighbors and loads are "B"(0.65) "C"(0.04) "D"(0.27) at that point you can process the likelihood of going from A to every one of those hubs as:

So you roll an arbitrary number from 0 to 1 and relying upon the likelihood of each connection you go to B, C or D. For this situation you are probably going to visit B from A.

$$A \Rightarrow B = \frac{0.65}{0.65 + .0\ .04 + .027}$$

$$A \Rightarrow C = \frac{0.04}{0.65 + .0\ .04 + .027}$$

$$A \Rightarrow D = \frac{0.27}{0.65 + .0.04 + .027}$$

One issue with this methodology is that you need to choose the length of each walk and the quantity of strolls to execute. There's a rich answer for this issue. Simply do a solitary huge irregular walk , at each progression with likelihood beta you visit a neighbor and with likelihood 1-beta you arbitrarily leap to any arbitrary hub, which is an approach to reenact the beginning another walk. Beta is ordinarily around 0.85.

As you execute your irregular walk add 1 to the score of a sentence each time you visit it.

Toward the end you can sort the sentences by the occasions you visited them and you have your sentences positioned. Simply take the top-n sentences as per the pattern in which as they show up in the content and you have a rundown.

With this methodology you have an approach to program TextRank without doing any math and without utilizing frameworks, you simply need your chart and a capacity to process the closeness between sentences.

What is behind:

What TextRank does is exceptionally basic: it discovers how comparative each sentence is to any remaining sentences in the content. The main sentence is the one that is generally like all the others, in light of this the closeness capacity should be arranged to the semantic of the sentence, cosine comparability dependent on a sack of words approach can function admirably and BM25/BM25+ work actually pleasantly for TextRank.

In the event that you separate words rather than sentences and follow a similar calculation, utilizing a comparability work between words then you can utilize TextRank to remove watchwords from the content, the thought is that the word that is generally like the wide range of various words is the main one. Separating stop-words is significant here.

Boundaries:

1. The content you are summing up.

2. The number of sentences to use in the outline.

3. Which closeness work use to think about sentences.

4. The length of your irregular walk.

5. Beta.

6. Best of luck!

Chapter 3

SYSTEM DEVELOPMENT

3.1 Analysis of various algorithms available:

Text Summarization focuses on a more concise representation of the document in text format with loosing minimal possible information .On a broad spectrum of various TS algo. , most of them are either Extractive or Abstractive in nature.

Extractive methods, try to produce the summary using the sentences that are already present in the document. The summary includes the sentences that found represent the most of the information and puts them together. Here some method to rank or weigh the importance of the sentences are implicit to every Extc. Method.

Abstractive methods attempt to improve the semantic between the sentences, or it may even produce some new ones.

The Abst. Methods are still a hot topic of research and new algo. of such class are bound to come up in future.

Our project aims to produce an extractive summary of the given source text

21

Extractive methods are usually performed in three stps:

(i) Create an intermediate representation of the original text;

(ii) Sentence scoring

(iii) Selecting a summary consisting of several sentences.

Diverse setting of the wellspring of text a few procedures may yield preferable outcome over different techniques, subsequently it is astute to characterize a couple of classes of text sources to address this issue. Extensively a content source can be arranged either as a news story, research article or a blog.

Out of these the wellsprings of writings our task would be more one-sided to deliver a more effective synopsis for news stories and websites than research papers.

3.2 Various computational approaches towards extractive text summarization:

This part is partitioned into:
(I) sentence scoring techniques depiction and how we execute it
(ii) the instrument that gives a simple mix of these techniques
(iii) the framework stream of exercises.

A. Scentence scoring strategies:

Fifteen of the most well known sentence scoring strategies are utilized here.
A short review of every one of them and their usage subtleties are introduced here. By and large, sentence scoring techniques are arranged by three classifications:
Word-based Scoring, Sentence-based Scoring and Graph-based Scoring. Notice that all administrations give as a yield a score somewhere in the range of 0 and 1 for each sentence. A few usage utilize a score standardization step.

1)Word-based Scoring: The primary strategies utilized for sentence scoring depended on word scoring. In such methodologies, each word gets a score and the heaviness of each sentence is the amount of all scores of its constituent words. The main word-based scoring strategies are recorded beneath. Lexical Similarity: It depends on the presumption that the significant sentences are recognized by solid chains; Upper Case: This strategy allots higher scores to words that contain at least one capitalized letters; Proper Noun: This technique guesses that sentences that contain a higher number of formal people, places or things are potentially more significant than others. Word Frequency: As the name of the technique sug-gests, the more every now and again a words happens in the content, the higher its score; TF/IDF: It utilizes TF/IDF recipe [4] to score sentences; Word Co-event: gauges the likelihood of two terms in a content to show up close by each otherin a specific request;

2)Sentence-based Scoring: This methodology examines the highlights of the sentence itself, for example, the presence of sign articulations. It was utilized without precedent for 1968 [5]. The main strategies that follow this thought are depicted below.Cue-Phrases: by and large, the sentences began by "in", "taking everything into account", "our

examination", "the paper portrays" and underlines, for example, "thebest", "the most significant", "as per the investigation", "essentially", "significant", "specifically", "scarcely", "outlandish" just as space explicit extra expressions terms can be acceptable pointers of critical substance of a book report; Sentence Position: The situation of the sentence all in all effects on its significance. For instance, the main sentences will in general come toward the start of an archive; Sentence Resemblance to the Title: Sentence resemblanceto the title is the jargon cover between this sentence and the report title; Sentence Centrality: Sentence centrality is the jargon cover between a sentence and different sentences in the record; Sentence Length: This element is utilized to punish sentences that are either excessively short or long; Sentence Inclusion of Numerical Data: Usually the sentence that contains mathematical information is a significant one and it is probably going to be remembered for the report synopsis.

3) Graph-based Scoring: In chart based strategies the score is produced by the relationship among sentences. At the point when a sentence alludes to another it produces a connection with a related

weight between them. The heaps are used to create the scores of a sentence. Text Rank: It removes the critical watchwords from a book chronicle and moreover chooses the substantialness of the "noteworthiness" of words inside the entire report by using an outline based model ;Bushy Path of the Node: The tough method of a center (sentence) on a guide is portrayed as the amount of associations interfacing it to various center points (sentences) on the guide; Aggregate Similarity: Instead of counting the amount of associations partner a center point (sentence) to various centers (Bushy Path), complete comparability sums the heaps (resemblances) of the associations. B.Combining Sentence Scoring Methods Two better methodologies for combining the sentence scoring techniques are proposed:

(i)By Ranking: Every assistance chooses the fundamental sentences and the client consolidates it by one way or another; and By Punctuation: The administration scores each sentence and returns one sentence with refreshed scores. To improve reusability and to ease administration launch, theTemplate Method configuration design was executed. Al lthe strategies actualized in this module broaden the sentence scoring unique class. There are four techniques in this class:

(i)sentence Scoring Ranking: A theoretical strategy to execute the sentence scoring technique. The yield is a string list with sentences proposed to be remembered for the synopsis;

(ii)sentence Scoring Punctuation:An dynamic strategy to execute the sentence scoring technique. The yield is a rundown of sentences with the technique scoring set;

(iii)template Sentence Scoring Ranking: The solid strategies which play out some pre and post preparing prior to calling the sentenceScoringRanking dynamic technique;

(iv)template Sentence Scoring Punctuation: The solid strategies which play out some pre and post preparing prior to calling the sentenceScoringPuntiation dynamic strategy. The initial two techniques typify the principle part of the administration. Subsequently, all administrations must execute them. The layout SentenceScoringRanking and templateSentenceScoringPuntuation techniques execute the Template Method. Moreover, the Factory Method plan pattern[6] is utilized to make cases of sentence scoring administrations. In the current case such method is utilized to start up any assistance. The client doesn't have to realize which class executes the administration. A case is made in response to popular demand to the industrial facility utilizing the technique name. For instance, to make an occurrence of the WordFrequency class, it is important to demand it to the processing plant by utilizing the string "word recurrence".

3.3 Experimental trials:

1.Implementing textrank calculation on GloVe word embeddings:

GloVe word embeddings are vector portrayal of words. These word embeddings will be utilized to make vectors for our sentences. We might have additionally utilized the Bag-of-Words or TF-IDF ways to deal with make highlights for our sentences, yet these strategies overlook the request for the words (and the quantity of highlights is generally beautiful large).We will utilize the pre-prepared Wikipedia 2014 + Gigaword 5 GloVe vectors

The subsequent stage is to discover similitudes between the sentences, and we will utilize the cosine closeness approach for this test. We should make a vacant similitude grid for this undertaking and populate it with cosine likenesses of the sentences.

We should initially characterize a zero network of measurements (n * n).We will instate this grid with cosine likeness scores of the sentences. Here,n is the quantity of sentences.

2.Implementing Textrank algoritm on Word frequency vectors:

This approach weights the important part of sentences and uses the same to form the summary. Different algorithm and techniques are used to define weights for the sentences and further rank them based on importance and similarity among each other.

Input document → sentences similarity → weight sentences → select sentences with higher rank.

The limited study is available for abstractive summarization as it requires a deeper understanding of the text as compared to the extractive approach.

Purely extractive summaries often times give better results compared to automatic abstractive summaries. This is because of the fact that abstractive summarization methods cope with problems such as semantic representation,

inference and natural language generation which is relatively harder than data-driven approaches such as sentence extraction.

There are many techniques available to generate extractive summarization. To keep it simple, I will be using an unsupervised learning approach to find the sentences similarity and rank them. One benefit of this will be, you don't need to train and build a model prior start using it for your project.

It's good to understand Cosine similarity to make the best use of code you are going to see. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Since we will be representing our sentences as the bunch of vectors, we can use it to find the similarity among sentences. Its measures cosine of the angle between vectors. Angle will be 0 if sentences are similar.

3.The Textrank algorithm:

Prior to beginning with the TextRank calculation, there's another calculation which we should get comfortable with – the PageRank calculation. Truth be told, this really propelled TextRank! PageRank is utilized essentially for positioning website pages in online indexed lists. We should rapidly comprehend the rudiments of this calculation with the assistance of a model.

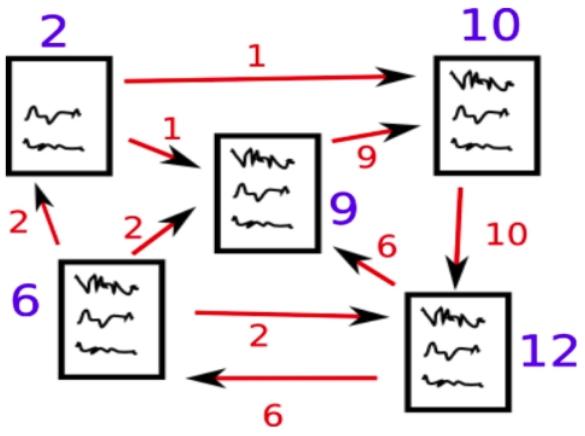


Fig. 2

Pagerank algorithm:A visual representation

Assume we have 4 site pages — w1, w2, w3, and w4. These pages contain joins highlighting each other. A few pages may have no connection – these are called hanging pages.

webpage	links
w1	[w4, w2]
w2	[w3, w1]
w3	[ ]
w4	[w1]

Fig 3

Site page w1 has joins coordinating to w2 and w4 w2 has joins for w3 and w1 w4 has interfaces just for the page w1 w3 has no connections and henceforth it will be known as a hanging page In request to rank these pages, we would need to register a score called the PageRank score. This score is the likelihood of a client visiting that page.

To catch the probabilities of clients exploring starting with one page then onto the next, we will make a square grid M, having n lines and n sections, where n is the quantity of pages.

M =	w1	w2	w3	w4
w1				
w2				
w3				
w4				

Fig 4

Every component of this framework signifies the likelihood of a client changing starting with one site page then onto the next. For instance, the featured cell beneath contains the likelihood of progress from w1 to w2.

M =	w1	w2	w3	w4
w1				
w2				
w3				
w4				

P(w1 to w2)

The initialization of the probabilities is explained in the steps below:

Probability of going from page i to j, i.e., M[i][j], is initialized with 1/(number of unique links in web page wi)

If there is no link between the page i and j, then the probability will be initialized with 0

If a user has landed on a dangling page, then it is assumed that he is equally likely to transition to any page. Hence, M[i][j] will be initialized with 1/(number of web pages)

M =	w1	w2	w3	w4
w1	0	0.5	0	0.5
w2	0.5	0	0.5	0
w3	0.25	0.25	0.25	0.25
w4	1	0	0	0

Fig 6

Hence, in our case, the matrix M will be initialized as follows

At last, the qualities in this network will be refreshed in an iterative design to show up at the site page rankings. TextRank Algorithm Let's comprehend the TextRank calculation, since we have a grip on PageRank. I have recorded the likenesses between these two calculations underneath: instead of pages, we use sentences Similarity between any two sentences is utilized as a comparable to the website page progress likelihood The

comparability scores are put away in a square lattice, like the network M utilized for PageRank TextRank is an extractive and unaided content outline strategy. We should investigate the progression of the TextRank calculation that we will be after: The initial step is link all the content contained in the articles Then split the content into singular sentences In the following stage, we will discover vector portrayal (word embeddings) for every single sentence

Similitudes between sentence vectors are then determined and put away in a lattice. The closeness network is then changed over into a diagram, with sentences as vertices and likeness scores as edges, for sentence rank calculation.Finally, a specific number of highest level sentences structure the last synopsis.

3.4 Mathematical aspect of the approach:

Cosine closeness is a measurement used to gauge how comparative the archives are regardless of their size. Numerically, it quantifies the cosine of the point between two vectors extended in a multi-dimensional space. The cosine closeness is beneficial in light of the fact that regardless of whether the two comparative reports are far separated by the Euclidean distance (because of the size of the archive), odds are they may in any case be situated nearer together. The more modest the point, higher the cosine comparability.

A normally utilized way to deal with coordinate comparative archives depends on tallying the greatest number of regular words between the reports. In any case, this methodology has a natural imperfection. That is, as the size of the archive builds, the quantity of basic words will in general increment regardless of whether the reports talk about various points. The cosine likeness beats this central blemish in the 'tally the-normal words' or Euclidean distance approach.

Cosine likeness is a measurement used to decide how comparative the reports are independent of their size. Numerically, it gauges the cosine of the point between two vectors extended in a multi-dimensional space. In this unique situation, the two vectors I am discussing are exhibits containing the word tallies of two records.

As a likeness metric, how does cosine closeness contrast from the quantity of basic words? At the point when plotted on a multi-dimensional space, where each measurement relates to a word in the report, the cosine closeness catches the direction (the point) of the records and not the extent. On the off chance that you need the size, process the Euclidean distance all things considered. The cosine closeness is invaluable on the grounds that regardless of whether the two comparative archives are far separated by the Euclidean distance in light of the size (like, the word 'cricket' seemed multiple times in a single report and multiple times in another) they could in any case have a more modest point between them. More modest the point, higher the similitude.

Illustration of cosine vector computation Let's guess you have 3 archives dependent on a few star cricket players – Sachin Tendulkar and Dhoni. Two of the records (A) and (B) are from the wikipedia pages on the particular players and the third report. (C) is a more modest piece from Dhoni's wikipedia page.

As should be obvious, each of the three archives are associated by a typical topic – the sport of Cricket. Our goal is to quantitatively gauge the comparability between the archives.

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

Doc Sachin: Wiki page on Sachin Tendulkar	
Dhoni	10
Cricket	50
Sachin	200

Doc Dhoni: Wiki page on Dhoni	
Dhoni	400
Cricket	100
Sachin	20

Doc Dhoni_Small: Subsection of wiki on Dhoni	
Dhoni	10
Cricket	5
Sachin	1

Document - Term Matrix (Word Counts)

Word Counts	"Dhoni"	"Cricket"	"Sachin"
Doc Sachin	10	50	200
Doc Dhoni	400	100	20
Doc Dhoni_Small	10	5	1

Similarity Metrics

Similarity or Distance Metrics	Total Common Words	Euclidean distance	Cosine Similarity
Doc Sachin & Doc Dhoni	10 + 50 + 10 = 70	432.4	0.15
Doc Dhoni & Doc Dhoni_Small	20 + 10 + 7 = 37	204.0	0.23
Doc Sachin & Doc Dhoni_Small	10 + 10 + 7 = 27	401.85	0.77

For ease of understanding, let's consider only the top 3 common words between the

records: 'Dhoni', 'Sachin' and 'Cricket'. You would anticipate Doc An and Doc C, that is the two reports on Dhoni would have a higher likeness over Doc An and Doc B, since, Doc C is basically a bit from Doc An itself. Be that as it may, in the event that we pass by the quantity of normal words, the two bigger records will have the most well-known words and consequently will be decided as generally comparable, which is actually what we need to stay away from.

The outcomes would be more consistent when we utilize the cosine closeness score to evaluate the comparability.

Projection of the archives in a 3-dimensional space, where each measurement is a recurrence tally of either: 'Sachin', 'Dhoni' or 'Cricket'. At the point when plotted on this space, the 3 archives would show up something like this.

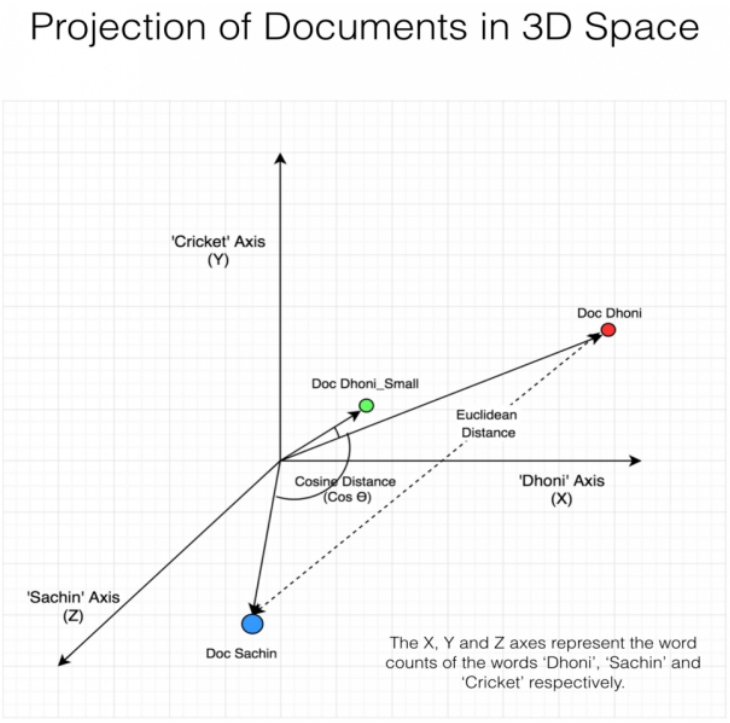


Fig 8

Au can see, Doc Dhoni_Small and the fundamental Doc Dhoni are situated nearer together in 3-D space, despite the fact that they are far separated by magnitiude. It ends up, the closer the records are by point, the higher is the Cosine Similarity (Cos theta).

As we include more words from the document, it's harder to visualize a higher dimensional space. But you can directly compute the cosine similarity using this math formula. 3.4 Statistical analysis and method of choice:

The two of above mentioned mehod in 3.3 only differ by the methodof calculating the

```
# Extract word vectors
word_embeddings = {}
f = open('glove.6B.100d.txt', encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    word_embeddings[word] = coefs
f.close()
```

Fig 9

vectors. We chose the second method (word frequency)method since it is more suitable for news article and less demanding in terms of resources as compared to GloVe word embeddings.

We have utilized python to execute the bit for tis venture.

1.Using GloVe word embeddings in python to make the vectors:

2.NLTK Libraries:

NLTK is a main stage for building Python projects to work with human language information. It gives simple to-utilize interfaces to more than 50 corpora and lexical assets, for example, WordNet, alongside a set-up of text handling libraries for characterization, tokenization, stemming, labeling, parsing, and semantic thinking, coverings for mechanical strength NLP libraries, and a functioning conversation gathering.

NLTK has been classified "an awesome instrument for instructing, and working in, computational etymology utilizing Python," and "an astonishing library to play with characteristic language."

Normal Language Processing with Python gives a handy prologue to programming for language handling. Composed by the makers of NLTK, it directs the peruser through the basics of composing Python programs, working with corpora, sorting text, breaking down

phonetic structure, and that's just the beginning. The online variant of the book has been refreshed for Python 3 and NLTK 3.

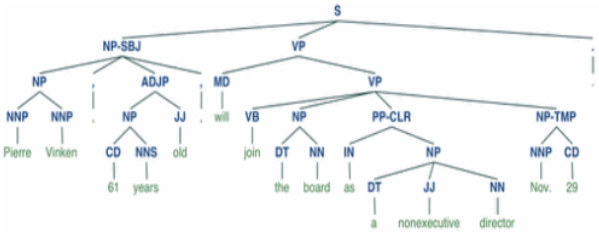
A couple of essential elements of nltk include:

a.Tokenize and label some content :

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
>>> from nltk.corpus import treebank
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> t.draw()
```

Fig 10

b.Display a parse tree :



5 NetworkX :

We have utilized python to execute the bit for tis venture.

1.Using GloVe word embeddings in python to make the vectors:

2.NLTK Libraries:

5

NLTK is a main stage for building Python projects to work with human language information. It gives simple to-utilize interfaces to more than 50 corpora and lexical assets, for example, WordNet, alongside a set-up of text handling libraries for characterization, tokenization, stemming, labeling, parsing, and semantic thinking,

coverings for mechanical strength NLP libraries, and a functioning conversation gathering.

NLTK has been classified "an awesome instrument for instructing, and working in, computational etymology utilizing Python," and "an astonishing library to play with characteristic language."

Normal Language Processing with Python gives a handy prologue to programming for language handling. Composed by the makers of NLTK, it directs the peruser through the basics of composing Python programs, working with corpora, sorting text, breaking down phonetic structure, and that's just the beginning. The online variant of the book has been refreshed for Python 3 and NLTK 3.

A couple of essential elements of nltk include:

a.Tokenize and label some content :

3.6 Matplotlib :

Matplotlib is a Python 2D plotting library which produces distribution quality figures in an assortment of printed version designs and intelligent conditions across stages. Matplotlib can be utilized in Python contents, the Python and IPython shells, the Jupyter journal, web application workers, and four graphical UI toolboxs.

Matplotlib attempts to make simple things simple and hard things conceivable. You can produce plots, histograms, power spectra, bar graphs, errorcharts, scatterplots, and so on, with only a couple lines of code. For models, see the example plots and thumbnail display.

For basic plotting the pyplot module gives a MATLAB-like interface, especially when joined with IPython. For the force client, you have full control of line styles, text style properties, tomahawks properties, and so forth, through an item situated interface or by means of a bunch of capacities natural to MATLAB clients

6.Language and Environment :

Conda:

Bundle, reliance and climate the executives for any language—Python, R, Ruby, Lua, Scala, Java, JavaScript, C/C++, FORTRAN, and that's only the tip of the iceberg. Conda is an open source bundle the executives framework and climate the board framework that sudden spikes in demand for Windows, macOS and Linux. Conda rapidly introduces, runs and updates bundles and their conditions. Conda effectively makes, spares, loads and switches between conditions on your neighborhood PC. It was made for Python programs, yet it can bundle and disperse programming for any language. Conda as a bundle director causes you discover and introduce bundles. On the off chance that you need a bundle that requires an alternate form of Python, you don't have to change to an alternate climate director, on the grounds that conda is likewise a climate administrator. With only a couple orders, you can set up an absolutely isolated climate to run that distinctive form of Python, while proceeding to run your typical adaptation of Python in your ordinary climate.

In its default design, conda can introduce and deal with the thousand bundles at repo.anaconda.com that are fabricated, assessed and kept up by Anaconda®. Conda can be joined with constant mix frameworks, for example, Travis CI and AppVeyor to give successive, mechanized testing of your code. The conda bundle and climate director is remembered for all adaptations of Anaconda and Miniconda. Conda is additionally remembered for Anaconda Enterprise, which gives nearby undertaking bundle and climate the executives for Python, R, Node.js, Java and other application stacks. Conda is additionally accessible on conda-produce, a network channel. You may likewise get conda on PyPI, however that approach may not be as modern.

3.7 Python:

Python is a deciphered, significant level, universally useful programming language. Made by Guido van Rossum and first delivered in 1991, Python's plan theory accentuates code lucidness with its striking utilization of huge whitespace. Its language builds and article arranged methodology mean to assist software engineers with composing, intelligent code for little and enormous scope ventures.

Python is powerfully composed and trash gathered. It bolsters various programming ideal models, including procedural, object-situated, and practical programming. Python is

frequently portrayed as a "batteries included" language because of its complete standard library.

3.8 Spyder IDE :

Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It offers a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

Beyond its many built-in features, its abilities can be extended even further via its plugin system and API. Furthermore, Spyder can also be used as a PyQt5 extension library, allowing developers to build upon its functionality and embed its components, such as the interactive console, in their own PyQt software.

3.9 Jupyter Notebook :

JupyterLab is an online intelligent advancement climate for Jupyter scratch pad, code, and information. JupyterLab is adaptable: design and orchestrate the UI to help a wide scope of work processes in information science, logical figuring, and AI. JupyterLab is extensible and measured: compose modules that add new parts and coordinate with existing ones.

The Jupyter Notebook is an open-source web application that permits you to make and share records that contain live code, conditions, representations and story text. Utilizations include: information cleaning and change, mathematical reproduction, measurable demonstrating, information representation, AI, and substantially more.

3.10 Web App

The frontend(User Interface) of this project is created using Angular 7.
Angular 7 is a Typescript framework that is used to create single page web applications and has many features bootstraped into it, like form validation, routing, route gaurd etc.

Angular uses a component based approach towars app building. The entities and their related business logic. Angular apps also have services that allow the app components to interact with each other and exchange data within the app or outside it.
Angular is based on typescript which is a scripting language and superset of javascript. Typescript offers a stricter typing discipline than javascript.

The Sass is being used for the UI. Sass is another scripting language that is interpreted to CSS

The App offers followinig use casses :

1. Filter news by country
2. Filter news by genre
3. Filter news by country and genre

The app uses news api to mitigate its core functions and serve news to the user
The App has following components :

1. Home Component
2. Navbar Component
3. Footer Component

The app also uses following services to interact within the components and the with the news api

1. Nav Service
2. News Service

3.11 Use cases in the Web App

3.11.1 Filter news by country

To show neews cards filtered according to region or country selected

311.2 Filter news by genre

To show news according to the category selected from the nav bar

3.11.3 Filter news by country and genre

To filter news according to country and category both by selecting country first and then category

3.12 Components of the web app

3.12.1 Home Component

This component displays news cards according to the selected filter applied by the user.
If no filters are applied by the user , then the default filters i.e. country : India and Category : general will be applied for the news.

3.12.2 NavBar Component

This component is responsible for adding the filter options and category options to the news app. These filter option are provided by the new.api and may change.

3.12.3 Footer Component

This component is a used to display the details of app and prominent news sources used by news.api.

3.13 Services used by the Web App

3.13.1 Nav service

This provides all filter options and categories to the NavBar component.
This service selects the filter provided and passes them to the news service

3.13.2 News service

API calls for the app made through this service and it helps in making the app modular and extensible. This service recieves the filter options and makes api calls

The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary

of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.’ The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough

we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.

The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.
The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary

of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough

we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document. The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.

Chapter 4

PERFORMANCE ANALYSIS

4.1 Analysis for the GloVe word embeddings method :

The GloVe strategy fundamentally utilizes the TF-IDF technique for making vectors.

a.Calculation of TF:

$$tf(t,d) = \frac{number\ of\ occurrences\ of\ term\ in\ document}{total\ number\ of\ all\ words\ in\ document}$$

The variable "TFVals" is ascertaining this equation. In the event that we run the sentence "Hi, my name is Brandon. Brandon. The elephant hops over the moon" through the term recurrence work, we'll get something that resembles this:

$$\overbrace{\text{Hello,}}^{0.125}\ \text{my}\ \overbrace{\text{name}}^{0.125}\ \text{is}\ \overbrace{\text{Brandon.}}^{0.375}\ \text{Brandon Brandon. The}\ \overbrace{\text{elephant}}^{0.125}\ \overbrace{\text{jumps}}^{0.125}\ \text{over the}\ \overbrace{\text{moon.}}^{0.125}$$

We have the term frequencies of words, but we want to calculate the most important sentences, not words. To do that, we go through every single sentence and see what words come up in that sentence which are in TFVals.

$$\overbrace{\text{Hello,}}^{0.125}\ \text{my}\ \overbrace{\text{name}}^{0.125}\ \text{is}\ \overbrace{\text{Brandon.}}^{0.375}\ \overbrace{\text{Brandon}}^{0.375}\ \overbrace{\text{Brandon.}}^{0.375}\ \text{The}\ \overbrace{\text{elephant}}^{0.125}\ \overbrace{\text{jumps}}^{0.125}\ \text{over the}\ \overbrace{\text{moon.}}^{0.125}$$

We simply need to add them all up and partition by the number of words we have. Since we're just including the TF estimations of relentless words, it's quite reasonable in the event that we partition by the number of constant words there are, rather than the number of words there are in a sentence. In the event that we don't partition by the number of words we have, long sentences have a bit of leeway over more limited ones.

$$\overbrace{\overbrace{\text{Hello,}}^{0.125}\ \text{my}\ \overbrace{\text{name}}^{0.125}\ \text{is}\ \overbrace{\text{Brandon.}}^{0.375}}^{\frac{0.125+0.125+0.375}{3}=0.278}\ \overbrace{\overbrace{\text{Brandon}}^{0.375}\ \overbrace{\text{Brandon.}}^{0.375}}^{\frac{0.375+0.375}{2}=0.375}\ \overbrace{\text{The}\ \overbrace{\text{elephant}}^{0.125}\ \overbrace{\text{jumps}}^{0.125}\ \text{over the}\ \overbrace{\text{moon.}}^{0.125}}^{\frac{0.125+0.125+0.125}{3}=0.125}$$

b. Computation of IDF:

Term recurrence is the way basic a word is, backwards report recurrence (IDF) is the way novel or uncommon a word is. The equation for IDF is:

$$idf(t,d) = log(\frac{how\ many\ times\ the\ term,\ t,\ appears.}{number\ of\ documents\ containing\ the\ term,\ t.})$$

IDF utilized over numerous reports, though TF is worked for one record. You can choose what a report is. In this article, each sentence is its own record.

The initial not many strides of IDF is equivalent to TF. We prettify the archive, include the words in the record and get all the exceptional words.

$$\overset{1}{\text{My}}\ \overset{1}{\text{name}}\ \text{is}\ \overset{1}{\text{Brandon.}}\ \text{I}\ \overset{1}{\text{love}}\ \overset{1}{\text{Brandon.}}\ \text{Is a}\ \overset{1}{\text{dog.}}$$

We're currently going to experience each and every word and tally how frequently that word shows up in each sentence and ascertain the IDF score utilizing the underneath equation.

$$log10(\frac{How\ many\ documents\ there\ are\ (sentences)}{How\ many\ times\ that\ word\ appears\ in\ all\ document})$$

$$IDF\ of\ "Brandon" = log10(\frac{3}{2}) = 0.176$$

Now we just do this for every non stop word.

$$\overset{0.477}{\text{My}}\ \overset{0.176}{\text{name}}\ \text{is}\ \overset{0.477}{\text{Brandon.}}\ \text{I}\ \overset{0.176}{\text{love}}\ \overset{0.477}{\text{Brandon.}}\ \text{Is a}\ \overset{0.477}{\text{dog}}.$$

4.2 Analysis for the WF method for findng vectors: Below is our code flow to generate summarize text:-

Input article → split into sentences → remove stop words → build a similarity matrix → generate rank based on matrix → pick top N sentences for summary.

Generate clean sentences:

```
def read_article(file_name):
    file = open(file_name, "r")
    filedata = file.readlines()
    article = filedata[0].split(". ")
    sentences = []

   for sentence in article:
     print(sentence)
     sentences.append(sentence.replace("[^a-zA-Z]", "
").split(" "))
     sentences.pop()

    return sentences
```

3

Similarity matrix

This is where we will be using cosine similarity to find similarity between sentences

```
def build_similarity_matrix(sentences, stop_words):
    # Create an empty similarity matrix
    similarity_matrix = np.zeros((len(sentences),
len(sentences)))

    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2: #ignore if both are same
sentences
                continue
            similarity_matrix[idx1][idx2] =
sentence_similarity(sentences[idx1], sentences[idx2],
stop_words)

return similarity_matrix
```

4.3 Results at various stages :

A. For the GloVe method :

```
In [2]: df = pd.read_csv("tennis_articles_v4.csv")
```

```
In [3]: df.head()
```

Out[3]:

	article_id	article_text	source
0	1	Maria Sharapova has basically no friends as te...	https://www.tennisworldusa.org/tennis/news/Mar...
1	2	BASEL, Switzerland (AP), Roger Federer advance...	http://www.tennis.com/pro-game/2018/10/copil-s...
2	3	Roger Federer has revealed that organisers of ...	https://scroll.in/field/899938/tennis-roger-fe...
3	4	Kei Nishikori will try to end his long losing ...	http://www.tennis.com/pro-game/2018/10/nishiko...
4	5	Federer, 37, first broke through on tour over ...	https://www.express.co.uk/sport/tennis/1036101...

```
In [4]: df['article_text'][0]
```

Out[4]:

```
"Maria Sharapova has basically no friends as tennis players on
the WTA Tour. The Russian player has no problems in openly spe
aking about it and in a recent interview she said: 'I don't re
ally hide any feelings too much. I think everyone knows this i
s my job here. When I'm on the courts or when I'm on the court
playing, I'm a competitor and I want to beat every single pers
on whether they're in the locker room or across the net.So I'm
not the one to strike up a conversation about the weather and
know that in the next few minutes I have to go and try to win
a tennis match. I'm a pretty competitive girl. I say my hello
s, but I'm not sending any players flowers as well. Uhm, I'm n
ot really friendly or close to many players. I have not a lot
of friends away from the courts.' When she said she is not rea
lly close to a lot of players, is that something strategic tha
t she is doing? Is it different on the men's tour than the wom
en's tour? 'No, not at all. I think just because you're in the
same sport doesn't mean that you have to be friends with every
one just because you're categorized, you're a tennis player, s
o you're going to get along with tennis players. I think every
person has different interests. I have friends that have compl
etely different jobs and interests, and I've met them in very
different parts of my life. I think everyone just thinks becau
se we're tennis players we should be the greatest of friends.
But ultimately tennis is just a very small part of what we do.
There are so many other things that we're interested in, that
we do.'"
```

In [10]:
```
sentences = []
for s in df['article_text']:
  sentences.append(sent_tokenize(s))

sentences = [y for x in sentences for y in x] # flatten list
```

In [11]:
```
sentences[:5]
```

Out[11]:
```
['Maria Sharapova has basically no friends as tennis players o
n the WTA Tour.',
 "The Russian player has no problems in openly speaking about
it and in a recent interview she said: 'I don't really hide an
y feelings too much.",
 'I think everyone knows this is my job here.',
 "When I'm on the courts or when I'm on the court playing, I'm
a competitor and I want to beat every single person whether th
ey're in the locker room or across the net.So I'm not the one
to strike up a conversation about the weather and know that in
the next few minutes I have to go and try to win a tennis matc
h.",
 "I'm a pretty competitive girl."]
```

In [24]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/bhaskar/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[24]:
```
True
```

In [25]:
```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
```

In [26]:
```
# function to remove stopwords
def remove_stopwords(sen):
    sen_new = " ".join([i for i in sen if i not in stop_words])
    return sen_new
```

In [ ]:
```
# remove stopwords from the sentences
clean_sentences = [remove_stopwords(r.split()) for r in clean_sentences]
```

In [27]:
```
# Extract word vectors
word_embeddings = {}
f = open('glove.6B.100d.txt', encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    word_embeddings[word] = coefs
f.close()
```

```
In [29]: clean_sentences
```

```
In [30]: sentence_vectors
```

```
Out[30]: [array([-0.00708815,  0.09256166,  0.63329273, -0.02311481,  0.01546605,
                  0.1853495 ,  0.09308354,  0.42582178, -0.24139106, -0.08799385,
                  0.24989085,  0.05680139,  0.1477317 ,  0.19214791, -0.11052089,
                 -0.13337198,  0.20583408,  0.11695171, -0.28442192,  0.33139458,
                  0.12599568,  0.1967534 ,  0.15575756,  0.5180779 ,  0.30478284,
                 -0.12746389,  0.04639504, -0.7776317 ,  0.37259623,  0.02550103,
                 -0.28453514,  0.3427335 ,  0.09285065,  0.07136673,  0.24223588,
                  0.153613  , -0.29634136,  0.39717883, -0.23219974, -0.1128767 ,
                 -0.32462594, -0.1468936 ,  0.35839075, -0.3026954 ,  0.16459417,
                 -0.15502267, -0.01870048, -0.2684455 ,  0.2881759 , -0.50023335,
                 -0.15034528, -0.2343842 ,  0.16917214,  0.63483965, -0.13028543,
                 -2.1654103 , -0.06345838,  0.08818175,  1.0153687 ,  0.8081116 ,
                 -0.23266864,  0.57686883, -0.20425446,  0.23006882,  0.33329207,
                 -0.12515107,  0.155755  ,  0.3699625 ,  0.16224106, -0.14049056,
                  0.0752349 , -0.14599869, -0.10349172, -0.28060302,  0.03092569,
                  0.11594792,  0.22164218,  0.07480332, -0.52976537, -0.07361726,
                  0.69293356, -0.20686361, -0.24250394,  0.08630013, -1.185478  ,
                 -0.38094005, -0.38220796,  0.11073201, -0.14750852, -0.1982723 ,
                 -0.2221683 , -0.07648158,  0.04440296,  0.2679849 , -0.13877201,
                  0.00610776, -0.39457384, -0.20842241,  0.50812364,  0.46988693],
                dtype=float32),
         array([-4.5618337e-02,  1.5819511e-01,  3.8475230e-01, -2.1156360e-01,
```

B .For WF method :

```
In [2]: file = open("tt.txt", "r")
        file2 = open("ww.txt","w")
        filedata = file.readlines()
        article = filedata[0].split(". ")
        """
        for w in article:
            file2.write(w)
            file2.write("\n")
        file2.close()
        print(filedata)
        print(article)
        print("\n--------------------\n")
        """
        sentences = []
        for sentence in article:
            print(sentence)
            sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
            sentences.append(re.sub('[^a-zA-Z]',' ', sentence).split(' '))
        sentences.pop()
        stopwords = stopwords.words('english')
```

```
I hope that my following explanation is helpful, if there is more information I can explain, please let me know via comm
ent or message and I will update the answer.Some Introduction:The "math" that goes into it is simple
A graph is a list of nodes (vertices) and thee connections (edges).TextRank uses the structure of the text and the known
parts of speech for words to assign a score to words that are keywords for the text
I don't think it takes much if any math to understand the concepts of the paper, but the terminology may be somewhat den
se the first time through
The algorithm gives more value to nodes with lots of connections, and gives more influence in steps to better connected
nodes, so it reinforces itself and eventually finds its stable score
The structure d the text is represented in the way the graph is made.The TextRank Algorithm:First, the words are assigne
```

```
In [5]: sent1=['For', 'years,', 'Facebook', 'gave', 'some', 'of', 'the', "world's", 'largest', 'technology', 'companies', 'more',
        sent2=['The', 'special', 'arrangements', 'are', 'detailed', 'in', 'hundreds', 'of', 'pages', 'of', 'Facebook', 'documents
        nltk.download("stopwords")
        stop_words = nltk.corpus.stopwords.words('english')
        if stop_words is None:
            stop_words = []
        sent1 = [w.lower() for w in sent1]
        sent2 = [w.lower() for w in sent2]

        all_words = list(set(sent1 + sent2))

        vector1 = [0] * len(all_words)
        vector2 = [0] * len(all_words)

        # build the vector for the first sentence
        for w in sent1:
            if w in stopwords:
                continue
            vector1[all_words.index(w)] += 1

        # build the vector for the second sentence
        for w in sent2:
            if w in stopwords:
                continue
            vector2[all_words.index(w)] += 1
        print(vector1)
        print(vector2)

        [0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
        1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
```

4.4 Comparison between the two method:

As referenced before , the TF-IDF approach includes more perplexing estimations when contrasted with the WF strategies. This overhead isn't just obvious regarding time devoured for producing the outcome yet additionally the memory devoured by the two portions while working on comparative datasets.

Henceforth we picked the WF approach for our venture.

Chapter 5

CONCLUSIONS

5.1 Conclusion:

The approach of WF - Cosine matrix scoring has generated good results on text data of any size (the size of the text data is measured using number of sentences in it). Altough we have used the top_n = 5 sentences as the hyperparameter for generating the summary of input text file but have also included a consolidated value of top_n = N/2 (where N is the size of input text file) to generate a more comprehensive summary of the document.

5.2 Future Scope:

The project is already being used in commercial world for summarizing big news articles into small headlines and RSS feeds. We are planning to implementing this kernel onto a front end and host it using a web service and create a small widget/application that automatically summarizes the results from the first few links that are generated when a user searches a query on some search engine,

5.3 Addition of extra features to web app :

The web app is still in a nascent stage and can use extra features to cater a wide audience. Following features are presently prototyped and will be added to the web app soon :

1. Addition of social share feature
2. Addition of email updates for selected filters
3. Addition of account based activity log for keeping record of selected filters and email preferences

6 Screenshots of the Web App

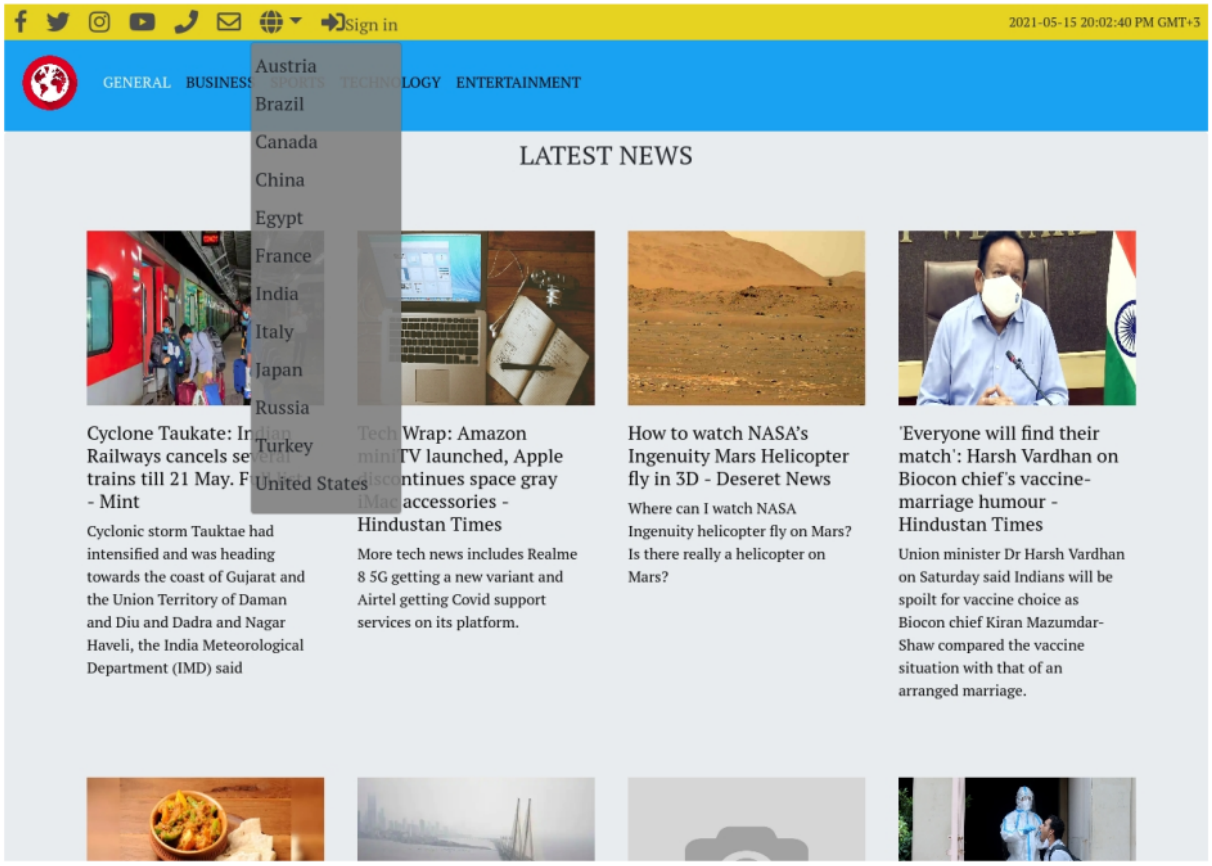


Fig. 11
Screenshot showing the
country filters

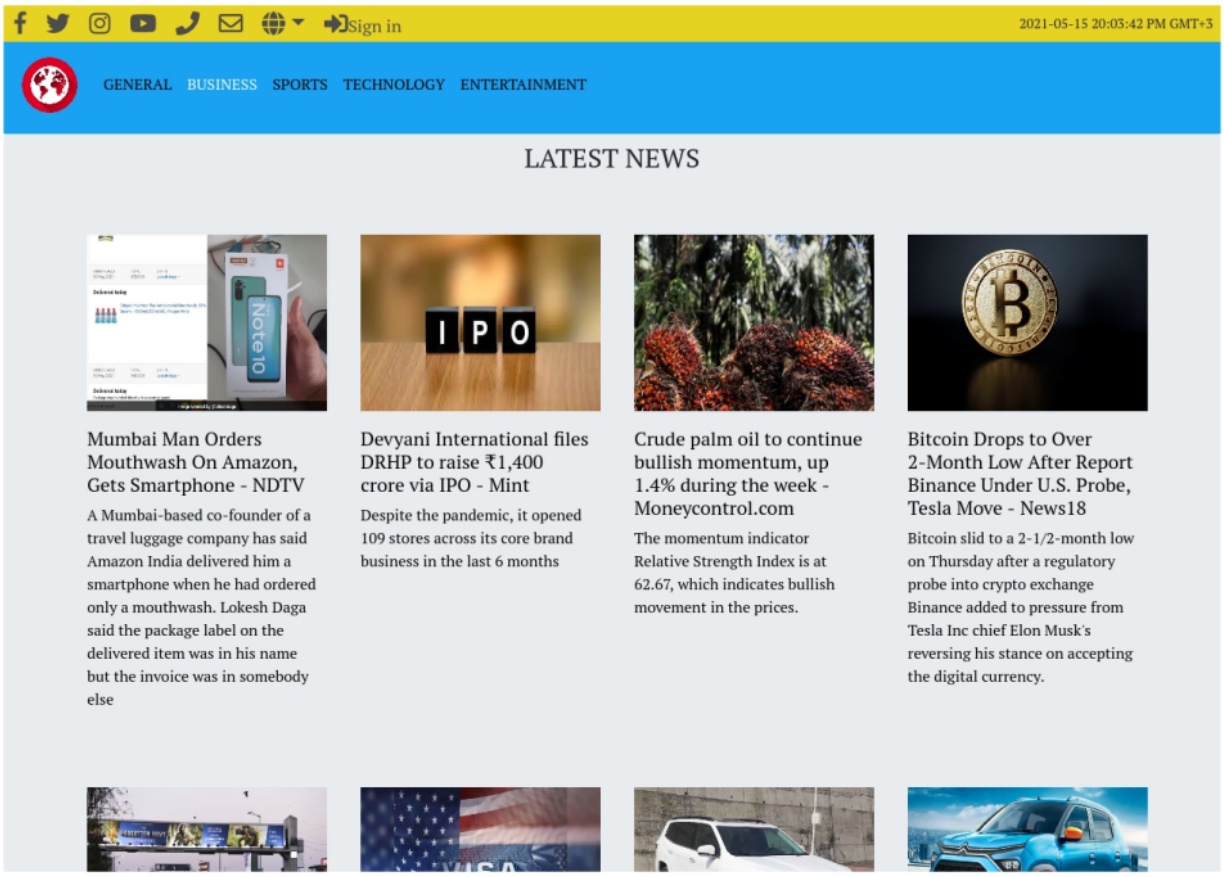


Fig. 12
Screenshot showing the
category filters

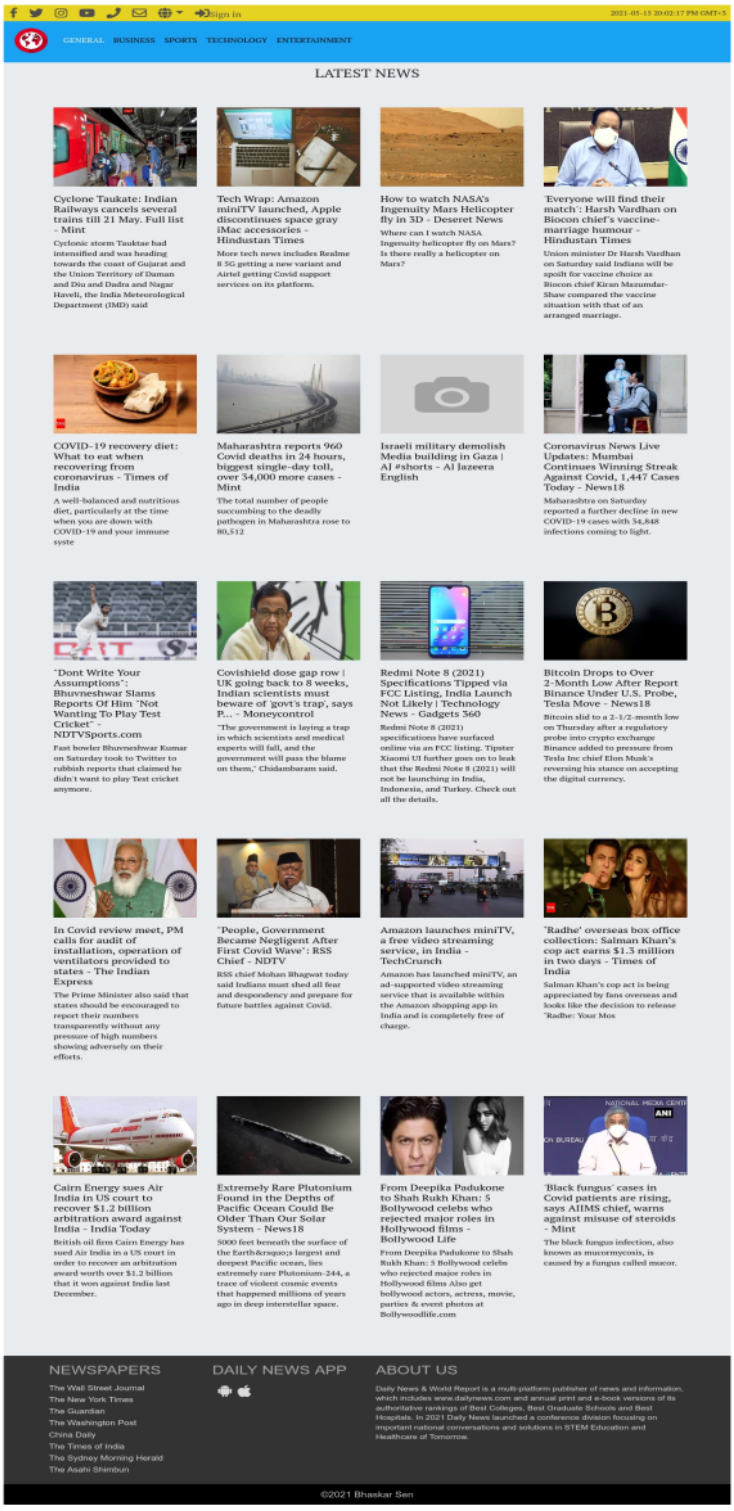


Fig. 13
Screenshot showing the default filters

bhaskar

ORIGINALITY REPORT

29%	20%	13%	16%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Rafael Ferreira, Frederico Freitas, Luciano de Souza Cabral, Rafael Dueire Lins et al. "A Context Based Text Summarization System", 2014 11th IAPR International Workshop on Document Analysis Systems, 2014 Publication	4%
2	Submitted to Institute of Technology, Nirma University Student Paper	3%
3	towardsdatascience.com Internet Source	3%
4	www.analyticsvidhya.com Internet Source	2%
5	Submitted to Asia Pacific Instutute of Information Technology Student Paper	2%
6	www.machinelearningplus.com Internet Source	2%
7	www.mecs-press.org Internet Source	1%

8	Submitted to The University of Buckingham Student Paper	1 %
9	skerritt.blog Internet Source	1 %
10	Submitted to Amity University Student Paper	1 %
11	Submitted to Ghana Technology University College Student Paper	1 %
12	Submitted to Guru Jambheshwar University of Science & Technology Student Paper	1 %
13	docs.conda.io Internet Source	1 %
14	link.springer.com Internet Source	1 %
15	Submitted to University of Westminster Student Paper	1 %
16	www.gcreddy.com Internet Source	<1 %
17	"Advances in Signal Processing and Intelligent Recognition Systems", Springer Science and Business Media LLC, 2021 Publication	<1 %
	Submitted to Kookmin University	

18	Student Paper	<1%
19	soumyadipnandi.wordpress.com Internet Source	<1%
20	Submitted to The British College Student Paper	<1%
21	www.coursehero.com Internet Source	<1%
22	"Query based Text Summarization", International Journal of Recent Technology and Engineering, 2019 Publication	<1%
23	Submitted to Southampton Solent University Student Paper	<1%
24	Submitted to Yaba College of Technology Student Paper	<1%
25	Rahim Khan, Yurong Qian, Sajid Naeem. "Extractive based Text Summarization Using KMeans and TF-IDF", International Journal of Information Engineering and Electronic Business, 2019 Publication	<1%
26	Prakhar Sethi, Sameer Sonawane, Saumitra Khanwalker, R. B. Keskar. "Automatic text summarization of news articles", 2017	<1%

International Conference on Big Data, IoT and Data Science (BID), 2017
Publication

27	Tanusree De, Debapriya Mukherjee. "Chapter 3 Explainable NLP: A Novel Methodology to Generate Human-Interpretable Explanation for Semantic Text Similarity", Springer Science and Business Media LLC, 2021 Publication	<1 %
28	www.mitpressjournals.org Internet Source	<1 %
29	"Soft Computing in Data Analytics", Springer Science and Business Media LLC, 2019 Publication	<1 %
30	medium.com Internet Source	<1 %
31	Daniel G. A. Smith, Doaa Altarawy, Lori A. Burns, Matthew Welborn et al. " The QCA project: An open - source platform to compute, organize, and share quantum chemistry data ", WIREs Computational Molecular Science, 2020 Publication	<1 %
32	Submitted to National University Of Science and Technology Student Paper	<1 %
33	machinelearningmastery.com	

	Internet Source	<1%
34	Submitted to Hofstra University Student Paper	<1%
35	epdf.tips Internet Source	<1%
36	www.newsweek.com Internet Source	<1%
37	www.ukessays.com Internet Source	<1%
38	Rafael Dueire Lins, Rafael Ferreira Mello, Steve Simske. "DocEng'19 Competition on Extractive Text Summarization", Proceedings of the ACM Symposium on Document Engineering 2019, 2019 Publication	<1%
39	digilib.stmik-banjarbaru.ac.id Internet Source	<1%
40	alive-directory.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

REFERENCES

[1] “Language Independent Extractive Summarization” by Rada Mihalcea Department of Computer Science and Engineering University of North Texas.

[2] “ClusterRank: A Graph Based Method for Meeting Summarization” by Nikhil Garg, Benoit Favre, Korbinian Reidhammer, Dilek Hakkani-Tur.

[3] “Graph-Based Text Summarization Using Modified TextRank” by Chirantana Mallick, Ajit Kumar Das, Madhurima Dutta, Asit Kumar Das and Apurba Sarkar.

[4] “Extractive based Text Summarization Using K-Means and TF-IDF” by Rahim Khan, Yurong Qian, Sajid Naeem School of Software, Xinjiang University, Urumqi 830008, China.

[5] “What is a simple but detailed explanation of Textrank?” answered by, Luis Argerich, Professor of Data Science at the University of Buenos Aires. (UBA).

[6]” A Context Based Text Summarization System” Rafael Ferreira , Frederico Freitas , Luciano de Souza Cabral , Rafael Dueire Lins , Rinaldo Lima ,Gabriel Franc , Steven J. Simske , and Luciano Favaro, Informatics Center, Federal University of Pernambuco, Recife, Pernambuco, Brazil Department of statistics and informatics, Federal Rural University of Pernambuco, Recife, Pernambuco, Brazil

[7] ”Automatic Text Summarization of News Articles” Prakhar Sethi , Sameer Sonawane , Saumitra Khanwalker , R. B. Keskar ,Department of Computer Science Engineering, Visvesvaraya National Institute of Technology, India

[8] ” AUTOMATED TEXT SUMMARIZATION AND THE SUMMARIST SYSTEM”

Eduard Hovy and Chin-Yew Lin ,Information Sciences Institute of the University of Southern California

[9]” An Extractive Approach for English Text Summarization”Kanchan D. Patil 1 , Sandip A. Patil 2 , Yogesh S. Deshmukh, Department of Information Technology, Sanjivani College of Engineering, Kopargaon, India”

Appendices

```
##the code:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
"""
import nltk

#for the english stopwords present in the nltk
from nltk.corpus import stopwords

#for the cosine_distance() function present in nltk.cluster.util
import numpy as np
from nltk.cluster.util import cosine_distance

#imports networkx 2.4.0
#backcompatibilty of this code lies till networkx 1.1.0
import networkx as nx

#for plotting undirected graph using cosine matrix
import matplotlib.pyplot as plt
#%%
#reads the orignal atricle from a txt file

def read_article(file_name):
    file = open(file_name, "r")
    filedata = file.readlines()

    #makes a list of strings that are sentences present in the article
    article = filedata[0].split(". ")
```

```
    sentences = []
    for sentence in article:
        print(sentence)
        #depending on your version of conda ,any one of the below lines would work fine
        #seprates the words in the string article[sentence]
        #1
        sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
        #2
        #sentences.append(re.sub('[^a-zA-Z]',' ', sentence).split(' '))

    #leaves the last
    sentences.pop()

    return sentences,article.size()


#%%
#returns cell value for cosine matrix
#iterates over the sentences[] list and calculates the cosine distance between them
#cosine_distance(u, v):
#"""
#Returns 1 minus the cosine of the angle between vectors v and u. This is
#equal to 1 - (u.v / |u||v|).
#"""
#return 1 - (numpy.dot(u, v) / (sqrt(numpy.dot(u, u)) * sqrt(numpy.dot(v, v))))

def sentence_similarity(sent1, sent2, stopwords=None):

    #use empty list if stopwords aren't present for given language
    if stopwords is None:
        stopwords = []
```

```
#convert the words to lowercase for nltk.stopwords[]
sent1 = [w.lower() for w in sent1]
sent2 = [w.lower() for w in sent2]

#remove redundant words
all_words = list(set(sent1 + sent2))

#initialize vector for sent1 and sent2
#a numpy array can also be used by making further alterations in the code
vector1 = [0] * len(all_words)
vector2 = [0] * len(all_words)

# build the vector for the first sentence
for w in sent1:

    #remove stopwords
    if w in stopwords:
        continue
    vector1[all_words.index(w)] += 1

# build the vector for the second sentence
for w in sent2:

    #remove stopwords
    if w in stopwords:
        continue
    vector2[all_words.index(w)] += 1

#cosine distance calculated according to above formula
return 1 - cosine_distance(vector1, vector2)
```

```
#%%
#building cosine similarity matrix

def build_similarity_matrix(sentences, stop_words):
    # Create an empty similarity matrix
    similarity_matrix = np.zeros((len(sentences), len(sentences)))

    #iterates over every element of list sentences[] to calculte cosine distance
    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):

            #ignore if both are same sentences
            if idx1 == idx2:
                continue
            similarity_matrix[idx1][idx2] = sentence_similarity(sentences[idx1], sentences[idx2],
stop_words)

    return similarity_matrix


#%%

def generate_summary(file_name, top_n=5):
    nltk.download("stopwords")
    stop_words = stopwords.words('english')
    summarize_text = []

    ## Step 1 - Read text anc split it
    sentences ,top_n_consolidated=  read_article(file_name)
    top_n=top_n_consolidated

    ## Step 2 - Generate Similary Martix across sentences
```

```
    sentence_similarity_martix = build_similarity_matrix(sentences, stop_words)

    ## Step 3 - Rank sentences in similarity martix
    #creates an undirected graph using the square matrix obtained form build_similarity_matrix()
    sentence_similarity_graph =
nx.from_numpy_array(sentence_similarity_martix,parallel_edges=True, create_using=None)
    #draw the unlabled graph
    nx.draw(sentence_similarity_graph)
    plt.draw()

    #converts the undirected graph to a directed graph by adding two dircted graph for each
undirected edge
    #writes weights for the sentences
    scores = nx.pagerank(sentence_similarity_graph)

    ## Step 4 - Sort the rank and pick top sentences
    ranked_sentence = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)
    print("Indexes of top ranked_sentence order are ", ranked_sentence)

    #adds top_n sentences to the summarized text
    for i in range(top_n):
      summarize_text.append(" ".join(ranked_sentence[i][1]))

    ## Step 5 - output the summarize text
    print("Summarize Text: \n", ". ".join(summarize_text))


#%%

# let's begin
generate_summary( "tt.txt", 5
```