Text Summarization in Marathi using TextRank Algorithm

Submitted in partial fulfillment of the requirements
of the degree of

B. E. Computer Engineering

By

Shravani Dhuri	**6**	**182026**
Clare Rebello	**7**	**182092**
Kate Rebello	**8**	**182093**
Nipun Mundada	**16**	**182075**

Guide:

Name of the Guide: Mrs. Pradnya Sawant
Designation: Assistant Professor

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)

University of Mumbai
2021-2022

CERTIFICATE

This is to certify that the project entitled **"Text Summarization in Marathi using TextRank Algorithm"** is a bonafide work of **"Shravani Dhuri"(06), "Clare Rebello"(07), "Kate Rebello"(08), "Nipun Mundada"(16)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of B.E. in Computer Engineering .

Ms. Pradnya Sawant
Guide

Project Report Approval for B.E.

This project report entitled "**Text Summarization in Marathi using TextRank Algorithm**" by (***Shravani Dhuri ,Clare Rebello , Kate Rebello, Nipun Mundada***) is approved for the degree of ***B.E. in Computer Engineering.***

Examiners

1.----------

2.----------

Date:

Place:

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Shravani Dhuri 06)

(Clare Rebello 07)

(Kate Rebello 08)

(Nipun Mundada 16)

Date:

Abstract

In this modern era, the amount of data or information is huge and important. We want our NLP model to perform precisely and accurately for every task. To develop a well-performing model, various exploratory data analysis techniques like removing stop-words, stemming and lemmatizing the word are performed. But in a situation where the amount of data or information is huge. For example, there is a huge amount of words present in any review, and one can't go thoroughly from all the reviews, and will be required to summarize the text in such a manner where one can get an overview of the information. The text rank algorithm came here to provide automated summarized information of huge, unorganized information. This is not the only task we can perform by the package. Instead of summarizing, we can extract keywords and rank the phrase, making a huge amount of information understandable in a very summarized and short way.

Contents

List of Figures

List of Abbreviations

Sr. No.	Abbreviation	Expanded form
i	NLP	Natural Language Processing
ii	SOV	Subject Object Verb
iii	TF-ISF	Term Frequency/ Inverse Sentence Frequency

Chapter 1
Introduction

Text Summarization is a way of condensing actual text into an abstract form that provides the same meaning and knowledge as provided by the actual text. It chooses the foremost informative neighborhood of text and forms summaries that reveal the foremost purpose of the given document. It yields a summary produced by a summarization system which allows readers to understand the content of the document instead of reading each and every individual sentence.

1.1 Description

Text Summarization is an important task in Natural Language Processing (NLP) applications like text classification, question answering, legal texts summarization, news summarization, and headline generation. In this project, a text summarization system is developed to summarize a text file given in Marathi using TextRank Algorithm.

1.2 Problem Formulation

In the big data era, there has been an explosion in the amount of text data from a variety of sources. This volume of text is an inestimable source of information and knowledge which needs to be effectively summarized to be useful. This increasing availability of documents has demanded exhaustive research in the NLP area for automatic text summarization. Automatic text summarization is the task of producing a concise and fluent summary without any human help while preserving the meaning of the original text document.

1.3 Motivation

Various automatic text summarization systems are accessible for several often used languages. Many of the summarizers are for other foreign languages. Moreover, technical documentation is usually minimal or may be absent. When it involves Indian languages, automatic summarization systems are limited. Little or no research and work has been done on text summarization for the Indian language Marathi (an Under-Resourced language).

1.4 Proposed Solution

We propose starting with extractive text summarization algorithms as a base to our work. In order to accomplish this, we plan on using existing algorithms such as TextRank for keyword extraction. To implement text summarization for the Marathi Language, we will follow three main steps :

- **Preprocessing**: Involves processing the raw text corpus by following processes such as tokenization and stopword removal, while maintaining the sentence structure (the result of this step is a collection of sentences).
- **Stemming and/or lemmatization**: This involves reducing the form of a word to its stem/lemma. To accomplish this, language-specific knowledge is required. For this we use a rule based Marathi Stemmer.
- **Sentence ranking**: After the document's text has been processed to a collection of sentences in a normal form (in other words, we have a normalized version of the sentences), the words and sentences are ranked based on selected features (for example, thematic term and position).

1.5 Scope of The Project

This system gives a clear concise summary in Marathi of the given input text file. The system is developed in Python with a frontend developed using python framework Streamlit. In future, feature extraction process, features like SOV (Subject Object Verb - Experimental) verification, TF-ISF (Term Frequency/ Inverse Sentence Frequency) can be used to make the summary more relevant and precise.

Chapter 2

Review of Literature

Text Summarization methods can be classified into extractive and abstractive summarization. An extractive summarization method consists of selecting important sentences, paragraphs etc. from the original document and concatenating them into shorter form. The importance of sentences is decided based on statistical and linguistic features of sentences. An Abstractive summarization [1] attempts to develop an understanding of the main concepts in a document and then express those concepts in clear natural language. It uses linguistic methods to examine and interpret the text and then to find the new concepts and expressions to best describe it by generating a new shorter text that conveys the most important information from the original text document. In this project we focus on novel techniques which are based on extractive text summarization methods.

TextRank is a graph based ranking model for text processing, and shows how it can be successfully used for natural language applications. Mihalcea and Tarau[2] evaluated two innovative unsupervised approaches for keyword and sentence extraction, and showed that the accuracy achieved by TextRank in these applications is competitive with that of previously proposed state-of-the-art algorithms. An important aspect of TextRank is that it does not require deep linguistic knowledge, nor domain or language specific annotated corpora, which makes it highly portable to other domains, genres, or languages.

Gaikwad's[3] rule based Marathi text summarization method where noun words based on a set of questions for each sentence is generated. Thereafter, each question is ranked according to their importance and top ranked questions are extracted to obtain their answers. The collection of answers of these questions are considered as the summary of the document.

2.1 Page Rank Algorithm

The graph-based ranking algorithm used to find the importance for the nodes i.e. vertex within the graph. When one vertex is connected to another one, it is basically casting recommendation for that other vertex. The higher the number of votes that are cast for a vertex, the higher the importance of the vertex.

Formally, let G= {V,E}be a directed graph with the set of vertices V and set of edges E, Where E is subset of V*V. For a given vertex Vi, Let In(Vi) be the set of vertices that point to it (predecessors), and let Out(Vi) be the set of vertices that vertex Vi points to (successors). The score of a vertex Vi is defined as follows :

$$S(Vi) = (1 - d) + d * \sum_{j \epsilon In(Vi)} \frac{1}{|Out(Vj)|} S(Vj)$$

Where

'V'= set of vertices,

'E'=set of edges,

V(in) = Set of incoming edges,

V(out) = Set of outgoing edges,

d = damping factor (default =0.85),

W = set of edge weights

For undirected graphs,V(in) = V(out)

To take care of a case when there is no linking for any node . The probability, at any step, that the person will continue is a damping factor d. It is usually set to .85

Chapter 3

System Analysis

Detailed analysis of the Text Summarization System in Marathi is performed. We specify the functional and non-functional (implicit and explicit) requirements of the system. We show the interaction of the system and the user is illustrated using use case diagrams.

3.1 Functional Requirements

Home Page:

1. User gives a text file written in Marathi as input to the system.
2. As soon as the text file is uploaded, the system is given the text in the input file and the user is given summarized text as output.

Results:

Summarized text in marathi is justified and displayed to the user.

3.2 Non Functional Requirements:

3.2.1 Performance Requirements

The project is web based and can run from any text editor. Performance will depend on the input file given.

3.2.2 Software Quality Attributes

All users of this system are not expected to have a technical background, hence the system must be easy to use.

3.3 Specific Requirements:

Hardware :

A simple computer(desktop/laptop) which can run a web browser and a python backend is sufficient. The project is software based and has no special hardware requirements.

Software :

The application must support major web browsers with convenient access to the backend.

1. Windows 7 or higher.
2. Web Browser: Google Chrome 1.0, Internet Explorer 4.0, Firefox 1.0, Safari 1.0
3. Python
4. Text Editor

3.4 Use-Case Diagrams and description

In this section, the different use case scenarios in which the users interact with the system and each other are discussed.

Scenario 1: Text Summarization for Marathi text file:

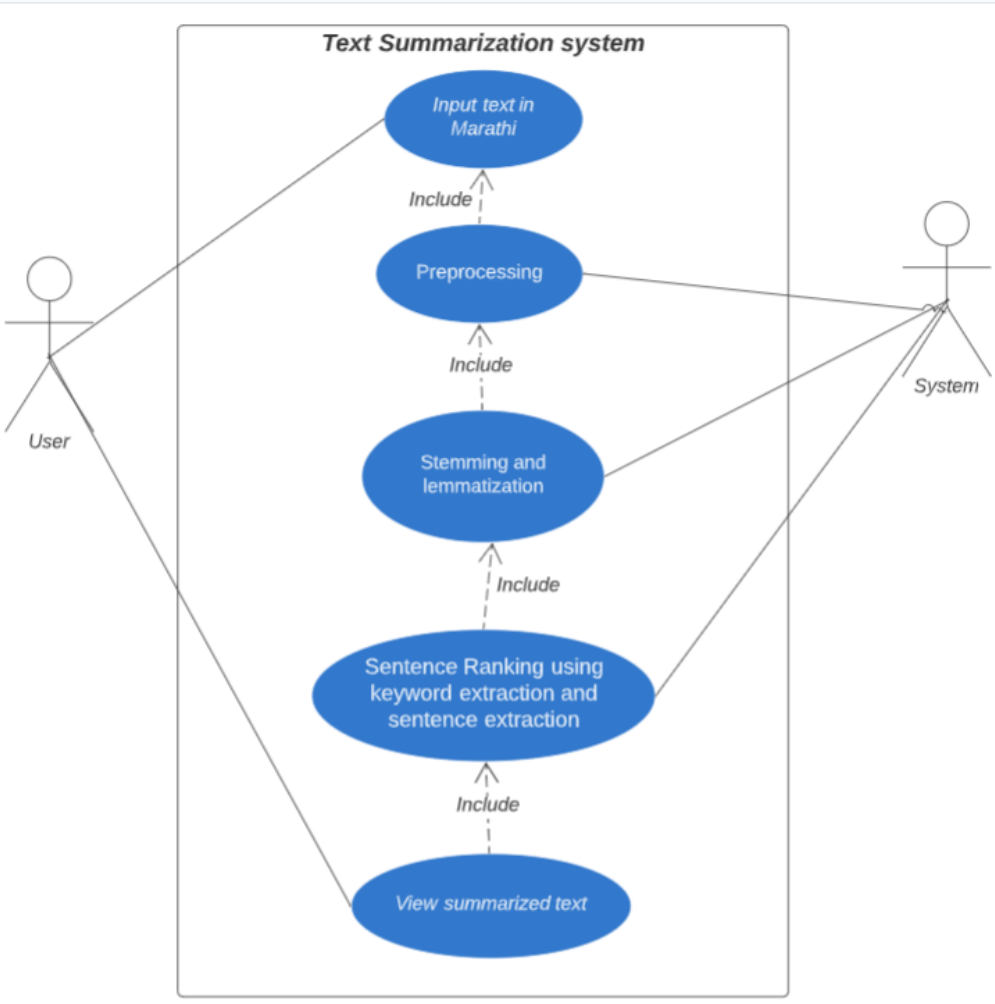


Fig. 3.4.1: Use Case Diagram for Text Summarization in Marathi

In Fig.3.4.1, a user uploads a Marathi text file in (UTF-8) into the upload area provided. As soon as a file is uploaded the system performs preprocessing steps on input text.After that stemming is performed and the sentences are ranked according to their importance and the result containing the summarized text is displayed to the user.

Use Case: Input text

Brief Description: The system will accept a single text file from the user.

Primary Actor: User

Main Flow: User provides input to the system.

Use Case: Preprocessing

Brief Description: System performs preprocessing such as tokenization and stopword removal

Primary Actor: System

Main Flow: After the user uploads a text file, it will perform preprocessing on the text present in the file using NLTK library.

Use Case: Stemming and/or Lemmatization

Brief Description: System performs Stemming on the given input text.

Primary Actor: System

Main Flow: After preprocessing is done by the system, it will perform stemming using the code written for Marathi Stemmer.

Use Case: Sentence ranking using keyword extraction and sentence extraction

Brief Description: System performs keyword/sentence extraction to rank the sentences in the given input text file.

Primary Actor: System

Main Flow: After stemming is done by the system, it will perform keyword extraction using TextRank algorithm and sentence ranking using Page rank algorithm.

Use Case: Gets output

Brief Description: The user can view the output on the screen

Primary Actor: User

Main Flow:

1.After the sentence ranking is done by the system, it will generate an output with the original and summarized text.
2.This output generated can be viewed by the users on the window.

Chapter 4

Analysis Modeling

Analysis modeling deals with modeling and representation of data. In this chapter, we design different models in which information, functions and the behavior of the system is defined and these are translated into the architecture.

4.1 Class Diagram and Activity Diagram

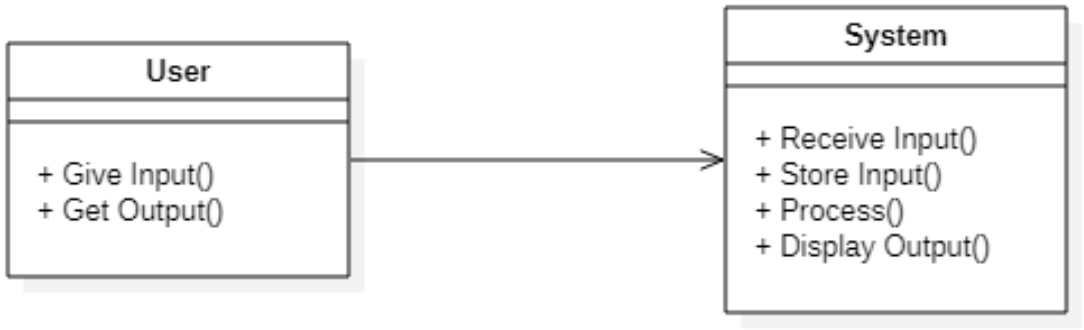
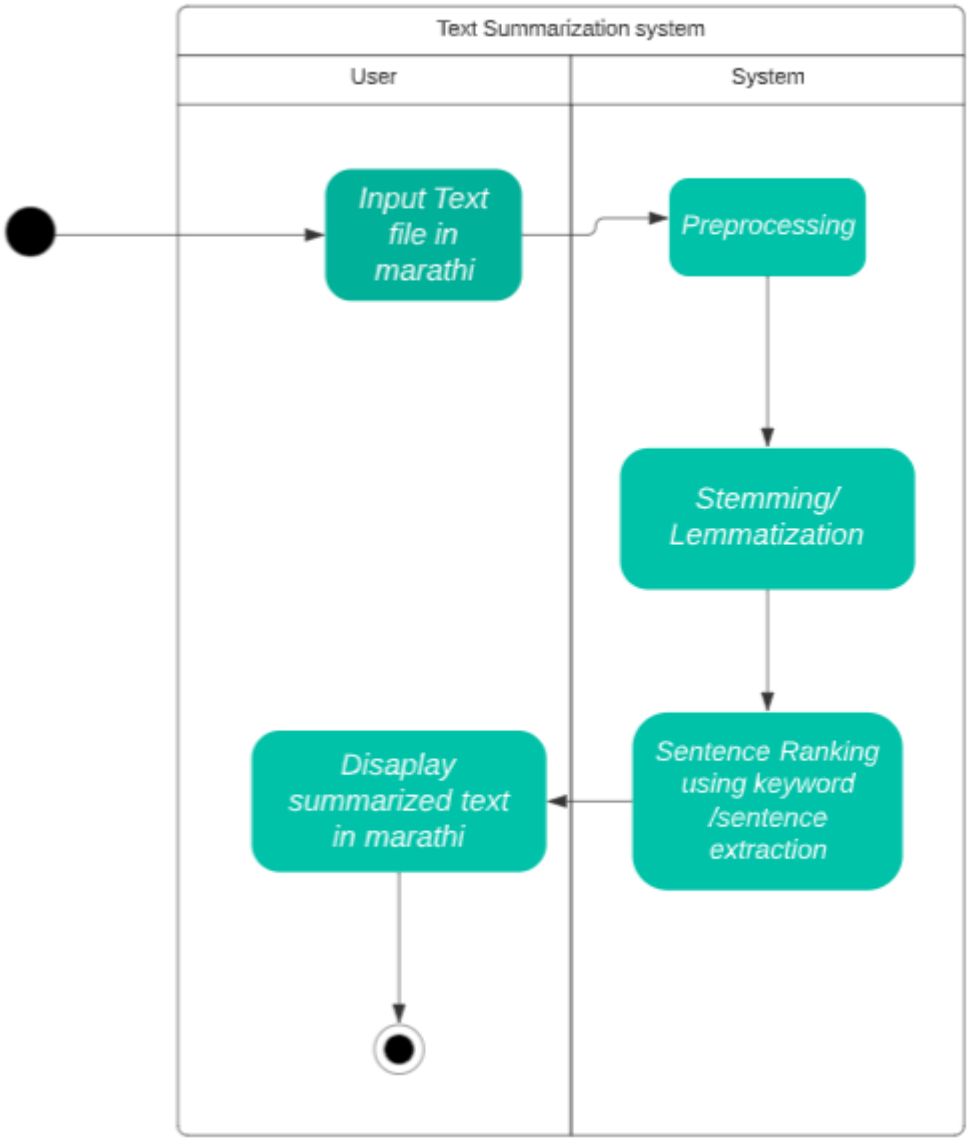


Fig 4.1.1: Class Diagram for Text Summarization System in Marathi

In this class diagram, Text Ranking on user input is performed and this is shown to the user.



4.1.2 Activity Diagram for Text Summarization System in Marathi

User inputs data which is preprocessed and fed to the keyword/sentence extractor. Text Summarization is done based on the sentence ranking and output is displayed to the user.

4.2 Functional Modeling

Data Flow Diagram

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on flow of information, where data comes from, where it goes and how it gets stored.

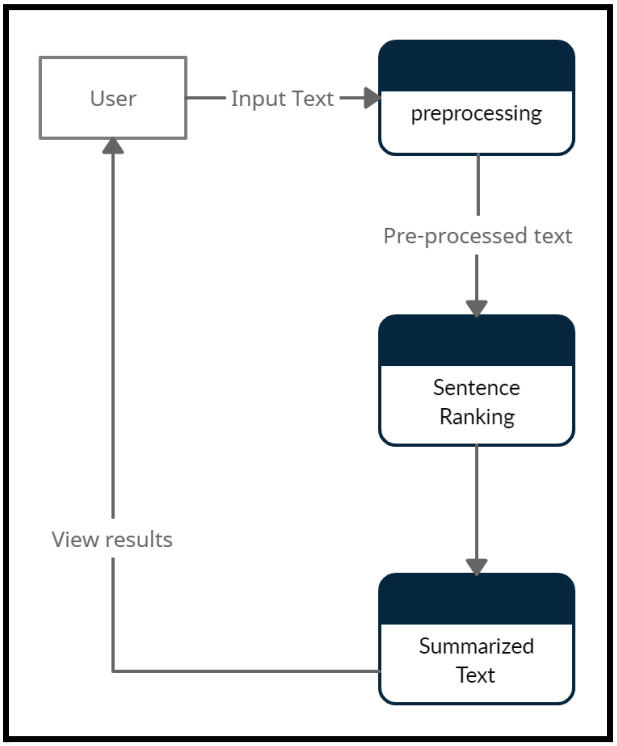


Fig.4.2.1 DFD (level -1)

The Marathi summarization system takes input from users and gives output summarized text based on the sentence ranks.

Chapter 5

Design

5.1 Architectural Design

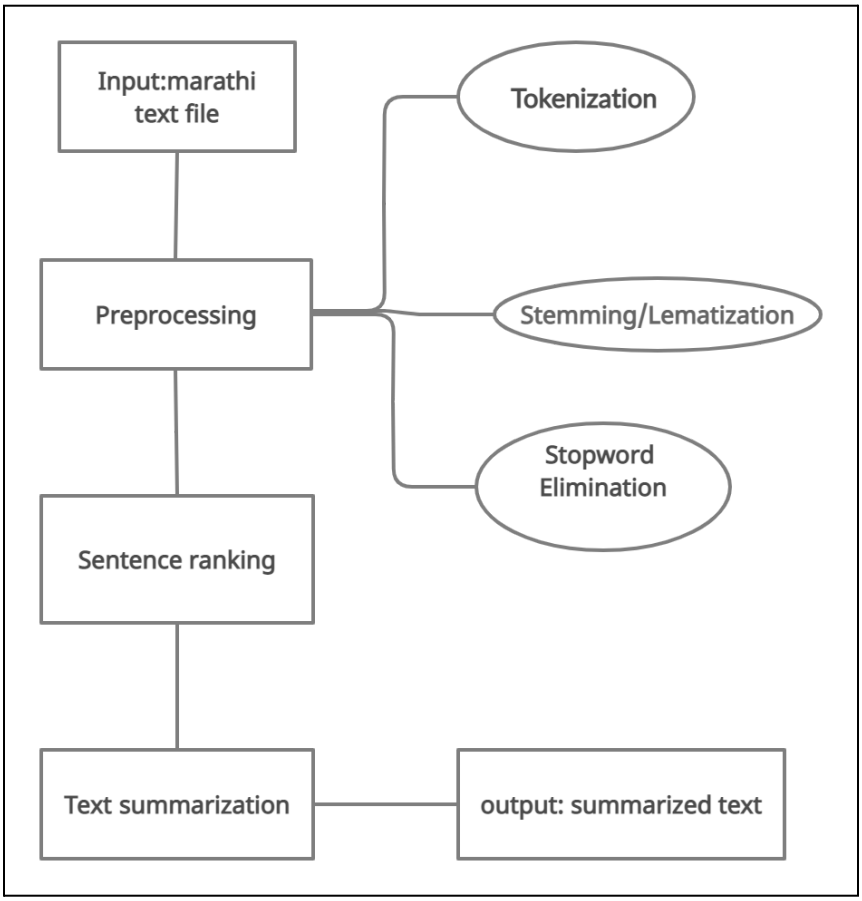


Fig. 5.1.1: Block Diagram for Text Summarization System in Marathi

In the diagram above, the architecture of the system is explained.

- Marathi text file is given as input to the system,which is first pre-processed wherein tokenization, stemming, stop-words removal is performed.
- After pre-processing, the next step is sentence ranking in which key-word or sentence is extracted and ranked using the Pagerank algorithm.
- After sentence ranking the summarized text is displayed to the user.

5.2 User Interface Design

The user interface design is created using streamlit, a python framework.Text file containing text in marathi is given as input and the system gives output the summarized text.

Fig 5.2.1:Accepting Input Text

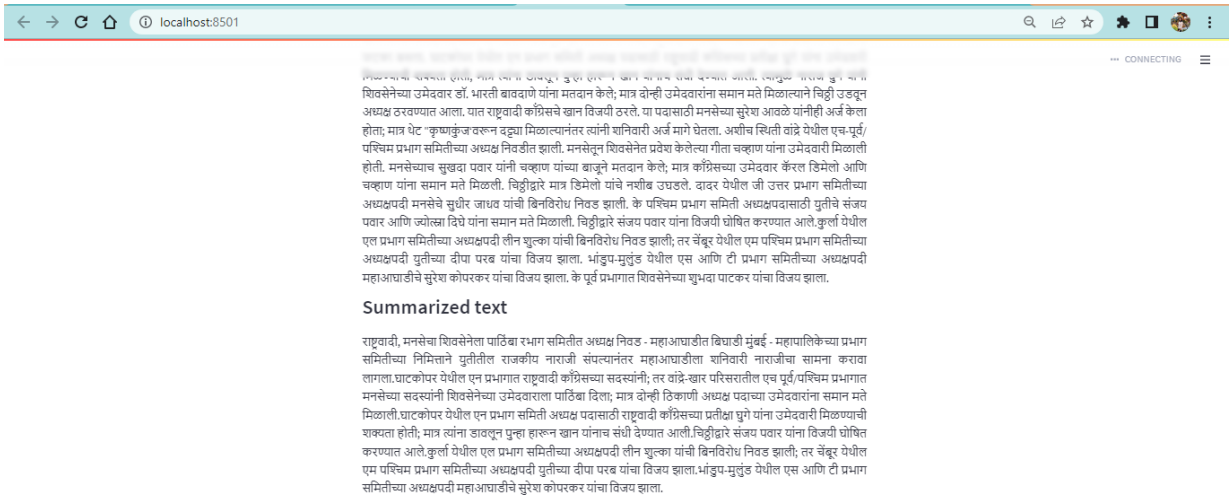


Fig 5.2.2: Summarized text output

Chapter 6

Implementation

In this project we have implement extraction based summarization method which has following three major steps:

A. Pre-processing

B. Stemming and/or lemmatization

C. Sentence ranking for Summary generation

6.1 Working of the project

6.1.1 Preprocessing

The pre-processing step includes tokenization and stop-word removal.Tokenization is the basic and important module of any NLP application.Tokenization is a way of separating a piece of text into smaller units called tokens.

```
Input Text: भांडुप येथील एस आणि टी प्रभाग समितीच्या अध्यक्षपदी महाआघाडीचे सुरेश कोपरकर यांचा विजय झाला.
Tokenized Text:
Token  0 : भांडुप
Token  1 : येथील
Token  2 : एस
Token  3 : आणि
Token  4 : टी
Token  5 : प्रभाग
Token  6 : समितीच्या
Token  7 : अध्यक्षपदी
Token  8 : महाआघाडीचे
Token  9 : सुरेश
Token  10 : कोपरकर
Token  11 : यांचा
Token  12 : विजय
Token  13 : झाला
```

For stop word removal, we have used the Marathi stop-word list text file downloadable from the internet. From the given document of Marathi text, remove the punctuation mark characters like ; , ——: ()[]{} space character, tab space and so on for finding individual Marathi words. Marathi language has some stop words that frequently occur in Marathi text. We eliminate these words from text; otherwise, sentences containing them can get importance unnecessarily.

```
Input Text: भांडुप येथील एस आणि टी प्रभाग समितीच्या अध्यक्षपदी महाआघाडीचे सुरेश कोपरकर यांचा विजय झाला.
Tokenized Text without stopwords: ['भांडुप', 'एस', 'टी', 'प्रभाग', 'समितीच्या', 'अध्यक्षपदी', 'महाआघाडीचे', 'सुरेश', 'कोपरकर', 'यांचा', 'विजय']
```

आहे	ते	कमी
या	असे	अनेक
आणि	होते	अधिक
व	केली	होणार
नाही	पण	म्हणाले
यानी	काही	याना
हे	केले	त्याची
तर	किवा	मी
आला	त्यामुळे	झाली
त	की	ता
येथे	न	टा

Fig 6.1.1: Marathi Stop Word Example

6.1.2 Stemming and/or Lemmatization

Using stemming, a word is split into its stem and affix. The design of a stemmer is language specific, and requires some significant linguistic expertise in the language. A stemmer algorithm involves removing suffixes using a list of frequent suffixes. If the word is found in Marathi noun or Marathi proper name list, its corresponding score is incremented by 1. If the word is not found in Marathi noun or Marathi proper name list then stemming is performed using any of the stemming rules.

```
Input Text: पाककडून पुन्हा एकदा शस्त्रसंधीचे उल्लंघन
Stemmed Text: ['पाककडून', 'पुन्हा', 'एकदा', 'शस्त्रसंधी', 'उल्लंघन']
```

Suffix	Marathi Word	Root Word
ला	रामला	राम
च	रामच	राम
चा	रामचा	राम
ची	रामची	राम
चे	रामचे	राम
च्या	रामच्या	राम

Fig 6.1.2: Marathi Stem Word Example

```
# coding=utf-8
from __future__ import print_function
from __future__ import unicode_literals
import collections
import copy
```

```
import io
import nltk
import re
from nltk.tokenize import sent_tokenize
nltk.download('punkt')
stopwords = set()
sentences = []
sentences_processing = []
sentence_dictionary = collections.defaultdict(dict)
stemWords = {}

def readStemWords():
    '''
        Reads the words from the stem words list and transforms the data
into usable format
     '''
    global stemWords
    with io.open("word_list_marathi.txt", encoding='utf-8') as textFile:
        index = 0
        for line in textFile:
            line = line.strip()
            if len(line) > 0:
                index += 1
                wordEndIndex = line.find(">")
                word = line[2:wordEndIndex]
                line = line[wordEndIndex + 1:]
                baseEndIndex = line.find("]")
                base = line[1:baseEndIndex].strip()
                line = line[baseEndIndex + 1:]
                stem = None
                if len(base) >= 0:
                    stemEndIndex = base.find('-')
                    if stemEndIndex > 0:
                        stem = base[:stemEndIndex]

                valid = line[line.find("(")+ 1: line.find(")")].strip()
                if valid == "0":
                    continue
                line = line[line.find("{") + 1: line.find("}")].strip()
                related = []
                if len(line) > 0:
                    split = line.split(",")
```

```
                    for s in split:
                        related.append(s[:s.find("|")])
                if stem == None and len(related) > 0:
                    stem = related[0]
                if stem != None:
                    stemWords[word] = {}
                    stemWords[word]["stem"] = stem
                    stemWords[word]["related"] = related

def tokenize(filename):
    '''
    Tokenizes the sentences and words
    :param filename: path of the file containing the text to be
summarized
    '''
    global sentences, sentences_processing, sentence_dictionary
    with io.open(filename, "r", encoding="utf-8") as inputFile:
        data = inputFile.read()
        inputFile.close()
    # data = filename.read().decode('utf-8')
    # filename.close()
    # data=filename
    sentences = sent_tokenize(data)
    sentences_processing = copy.deepcopy(sentences)
    counter = 0
    for sentence in sentences_processing:
        sentence = sentence[:-1]
        sentence = re.sub(',|\.|-|\(|\)', ' ', sentence)
        tokens = sentence.strip().split()
        actualTokens = removeStopWords(tokens)
        stemmedTokens = stemmerMarathi(actualTokens)
        sentence_dictionary[counter] = stemmedTokens
        counter += 1

def readStopWords():
    '''
    Reads the stopwords from the file
    '''
    with io.open("stopwords.txt", encoding='utf-8') as textFile:
        for line in textFile:
            words = line.lower().strip()
            stopwords.add(words)
```

```
        textFile.close()


def removeStopWords(wordlist):
    '''
    Removes the stopwords from the sentences
    :param wordlist: list of stopwords
    '''
    newlist = []
    for word in wordlist:
        if word not in stopwords:
            newlist.append(word)
    return newlist


def removeCase(word):
    '''
    :param word: word to be reduced its stem
    :return: stem of the word
    '''
    word_length = len(word) - 1
    if word_length > 5:
        suffix = "शया"
        if word.endswith(suffix):
            return word[:-len(suffix)]

    if word_length > 4:
        suffix = "शे"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "शी"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "चा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ची"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "चे"
        if word.endswith(suffix):
            return word[:-len(suffix)]
```

```
        suffix = "हून"
        if word.endswith(suffix):
            return word[:-len(suffix)]

    if word_length > 3:
        suffix = "नो"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "तो"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ने"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "नी"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ही"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ते"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "या"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ला"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ना"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ऊण"
        if word.endswith(suffix):
            return word[:-len(suffix)]

    if word_length > 2:
        suffix = " े"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ी"
        if word.endswith(suffix):
```

```
            return word[:-len(suffix)]
        suffix = "स"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ल"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "त"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "म"
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word

def removeNoGender(word):
    global stemWords
    orig = word
    if word in stemWords:
        return stemWords[word]["stem"]
    word_length = len(word) - 1

    if word_length > 5:
        suffix = " ुरडा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
    if word_length > 4:
        suffix = "ढा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
    if word_length > 3:
        suffix = "रु"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "डे"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ती"
        if word.endswith(suffix):
```

```
            return word[:-len(suffix)]
        suffix = " ान"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ीण"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "डा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "डी"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "गा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ला"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ळा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "या"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "वा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ये"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "वे"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ती"
        if word.endswith(suffix):
            return word[:-len(suffix)]
    if word_length > 2:
        suffix = "अ"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " े"
```

```
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "ि "
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ु"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ो"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ै"
        if word.endswith(suffix):
            return word[:-len(suffix)]

        suffix = " ा"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ी"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = " ू"
        if word.endswith(suffix):
            return word[:-len(suffix)]
        suffix = "त"
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word
def stemmerMarathi(words):
    return [removeNoGender(removeCase(word)) for word in words]
def cleanText(filename):
    '''
        Tokenize, Remove stopwords and reduce the words to their stem
    :param filename: path of file to be preprocessed
    '''
    global sentence_dictionary, sentences
    readStopWords()
    tokenize(filename)
    size = 0
    for i in range(0, len(sentence_dictionary)):
        size += len(sentence_dictionary[i])
```

```
    sentence_dictionary = {key: value for key, value in
sentence_dictionary.items() if len(value)>0}
    return sentence_dictionary, sentences, size
readStemWords()
```

6.1.3 Sentence Ranking for Summary Generation

After an input document is formatted and stemmed, the document is divided into a collection of sentences and the sentences are ranked based on two important features: thematic term and position.

To rank Marathi texts, we have to create a graph having the Marathi words, and interconnect those words or other text entities with meaningful relations. In graph text units of various sizes and characteristics can be included as vertices in the graph, e.g. words, collocations, entire sentences, or others .

Graph algorithms for ranking in Marathi follows the following main steps:

1. Identify text units that best define the task and add them as vertices in the graph.

2. Identify relations that connect such text units, and use these relations to draw edges between vertices in the graph. Edges can be directed or undirected, weighted or unweight.

3. Iterate the graph-based ranking algorithm until convergence.

4. Sort vertices based on their final score. Use the values attached to each vertex for ranking/selection decisions.

We implemented two natural language processing tasks involving ranking of text units:

(1) A keyword extraction task, consisting of the selection of keyphrases representative for a given text;

(2) A sentence extraction task, consisting of the identification of the most “important” sentences in a text, which can be used to build extractive summaries.

Keyphrase Extraction

This method is used to automatically identify text as a set of words which best describe the document. Such keywords can be useful entries for building an automatic index for a document collection, it is useful to classify a text to generate the summary. To apply Text-rank, we first need to build a graph associated with the text, where the graph vertices are representative of the units to be ranked. The co-occurrence relation used for keyword extraction is used here.With the help of these keywords, a graph is created to rank the sentences and generate summary.

```
Input Text
राष्ट्रवादी, मनसेचा शिवसेनेला पाठिंबा रभाग समितीत अध्यक्ष निवड - महाआघाडीत बिघाडी मुंबई - महापालिकेच्या प्रभाग समितीच्या निमित्ताने युतीतील राजकीय नाराजी संपल्यानंतर महाआघाडीला शनिवारी नाराजीचा सामना करावा लागला. घाटकोपर येथील एन प्रभागात राष्ट्रवादी काँग्रेसच्या सदस्यांनी; तर वांद्रे-खार परिसरातील एच पूर्व/पश्चिम प्रभागात मनसेच्या सदस्यांनी शिवसेनेच्या उमेदवाराला पाठिंबा दिला; मात्र दोन्ही ठिकाणी अध्यक्ष पदाच्या उमेदवारांना समान मते मिळाली. अखेर चिठ्ठ्यांद्वारे दोन्ही ठिकाणी महाआघाडीच्या उमेदवाराचे नशीब उघडले. पालिकेच्या आठ प्रभाग समित्यांच्या अध्यक्षपदासाठी शनिवारी निवडणूक झाली. यात काँग्रेस, राष्ट्रवादी काँग्रेस, मनसेच्या महाआघाडीला नाराजीचा फटका बसला. घाटकोपर येथील एन प्रभाग समिती अध्यक्ष पदासाठी राष्ट्रवादी काँग्रेसच्या प्रतीक्षा घुगे यांना उमेदवारी मिळण्याची शक्यता होती; मात्र त्यांना डावलून पुन्हा हारून खान यांनाच संधी देण्यात आली. त्यामुळे नाराज घुगे यांनी शिवसेनेच्या उमेदवार डॉ. भारती बावदाणे यांना मतदान केले; मात्र दोन्ही उमेदवारांना समान मते मिळाल्याने चिठ्ठी उडवून अध्यक्ष ठरवण्यात आला. यात राष्ट्रवादी काँग्रेसचे खान विजयी ठरले. या पदासाठी मनसेच्या सुरेश आवळे यांनीही अर्ज केला होता; मात्र थेट "कृष्णकुंज'वरून दट्ट्या मिळाल्यानंतर त्यांनी शनिवारी अर्ज मागे घेतला. अशीच स्थिती वांद्रे येथील एच-पूर्व/पश्चिम प्रभाग समितीच्या अध्यक्ष निवडीत झाली. मनसेतून शिवसेनेत प्रवेश केलेल्या गीता चव्हाण यांना उमेदवारी मिळाली होती. मनसेच्याच सुखदा पवार यांनी चव्हाण यांच्या बाजूने मतदान केले; मात्र काँग्रेसच्या उमेदवार कॅरल डिमेलो आणि चव्हाण यांना समान मते मिळली. चिठ्ठीद्वारे मात्र डिमेलो यांचे नशीब उघडले. दादर येथील जी उत्तर प्रभाग समितीच्या अध्यक्षपदी मनसेचे सुधीर जाधव यांची बिनविरोध निवड झाली. के पश्चिम प्रभाग समिती अध्यक्षपदासाठी युतीचे संजय पवार आणि ज्योत्स्ना दिघे यांना समान मते मिळाली. चिठ्ठीद्वारे संजय पवार यांना विजयी घोषित करण्यात आले.कुर्ला येथील एल प्रभाग समितीच्या अध्यक्षपदी लीन शुल्का यांची बिनविरोध निवड झाली; तर चेंबूर येथील एम पश्चिम प्रभाग समितीच्या अध्यक्षपदी युतीच्या दीपा परब यांचा विजय झाला. भांडुप-मुलुंड येथील एस आणि टी प्रभाग समितीच्या अध्यक्षपदी महाआघाडीचे सुरेश कोपरकर यांचा विजय झाला. के पूर्व प्रभागात शिवसेनेच्या शुभदा पाटकर यांचा विजय झाला.
Keyword Extracted:
['राष्ट्रवादी', 'प्रभाग', 'अध्यक्ष', 'समितीच्', 'महाआघा', 'अध्यक्षपदी', 'मनसे', 'प्रभा', 'विजय', 'शिवसेनेच्', 'पाठिंबा', 'निवड', 'यांचा', 'शिवसेनेच्', 'शनिवारी', 'काँग्रेसच्', 'दोन्',
'मनसेच्', 'युती', 'नाराजी', 'सुरेश', 'पवार', 'समान', 'चव्हाण']
```

To avoid growth of the graph size by adding all possible combinations of sequences consisting of more than one lexical unit (ngrams), we have used only single words for addition to the graph, with multi-word keywords being finally reconstructed in the post-processing phase. Later on , all lexical units that pass the syntactic filter for Marathi language are added to the graph, and an edge is added between those lexical units that co-occur within a window of Marathi words. After the graph is constructed, the score associated with each vertex containing a Marathi word is set to an initial value of 1, and the ranking algorithm is run on the graph for several iterations until it converges – usually for 20-30 iterations, at a threshold of 0.0001.Once a final score is obtained for each vertex in the graph, vertices are sorted in reversed order of their score, and the top vertices in the ranking are retained for post-processing.

```
# coding=utf-8
from __future__ import print_function
import collections
import io
import math
import operator
import sys
import networkx as nx
from preprocess import cleanText

window = 10
numberofSentences = 6
nodeHash = {}
textRank = {}
sentenceDictionary = collections.defaultdict(dict)
size = 0
sentences = []


def generatepositionaldistribution():
    '''
```

```
        Creates a weighted positional distribution of sentence scores
based on their position in the text corpus
    '''
    global nodeHash, sentenceDictionary, sentences, size
    positional_dictionary = collections.defaultdict(dict)
    count = 0
    for i in sentenceDictionary.keys():
        for j in range(0, len(sentenceDictionary[i])):
            count += 1
            position = float(count) / (float(size) + 1.0)
                    positional_dictionary[i][j] = 1.0 / (math.pi *
math.sqrt(position * (1 - position)))
            word = sentenceDictionary[i][j]
            if word in nodeHash:
                if nodeHash[word] < positional_dictionary[i][j]:
                    nodeHash[word] = positional_dictionary[i][j]
            else:
                nodeHash[word] = positional_dictionary[i][j]

def textrank():
    '''
        Generates a graph based ranking model for the tokens
       :return: Keyphrases that are most relevant for generating the
summary.
    '''
    global sentenceDictionary, nodeHash, textRank
    graph = nx.Graph()
    graph.add_nodes_from(nodeHash.keys())
    for i in sentenceDictionary.keys():
        for j in range(0, len(sentenceDictionary[i])):
            current_word = sentenceDictionary[i][j]
            next_words = sentenceDictionary[i][j + 1:j + window]
            for word in next_words:
                                    graph.add_edge(current_word, word,
weight=(nodeHash[current_word] + nodeHash[word]) / 2)
    textRank = nx.pagerank(graph, weight='weight')
    keyphrases = sorted(textRank, key=textRank.get, reverse=True)[:n]
    return keyphrases

def summarize(filePath, keyphrases, numberofSentences):
    '''
        Generates the summary and writes the summary to the file.
```

```
    :param filePath: path of file to be used for summarization.
    :param keyphrases: Extracted keyphrases
    :param numberofSentences: Number of sentences needed as a summary
    :output: Writes the summary to the file
    '''
    global textRank, sentenceDictionary, sentences
    sentenceScore = {}
    for i in sentenceDictionary.keys():
        position = float(i + 1) / (float(len(sentences)) + 1.0)
         positionalFeatureWeight = 1.0 / (math.pi * math.sqrt(position *
(1.0 - position)))
        sumKeyPhrases = 0.0
        for keyphrase in keyphrases:
            if keyphrase in sentenceDictionary[i]:
                sumKeyPhrases += textRank[keyphrase]
        sentenceScore[i] = sumKeyPhrases * positionalFeatureWeight
               sortedSentenceScores    =    sorted(sentenceScore.items(),
key=operator.itemgetter(1), reverse=True)[:numberofSentences]
                sortedSentenceScores    =    sorted(sortedSentenceScores,
key=operator.itemgetter(0), reverse=False)
    print("\nSummary: ")
    summary = []
    arr=[]
    # for keyphrase in keyphrases:
    #     print(keyphrase)
    # print(keyphrases)

    for i in range(0, len(sortedSentenceScores)):
        arr.append(sentences[sortedSentenceScores[i][0]])
    s="".join(arr)
    # print(s)
    return(s)

def process(arg1):
    '''
    :param arg1: path to the file containing the text to be summarized
    :param arg2: Number of sentences to be extracted as summary
    :param arg3: size of the window to be used in the co-occurance
    '''
    arg2=5
    arg3=6
```

```
    global window, n, numberofSentences, textRank, sentenceDictionary,
size, sentences
    if arg1 != None and arg2 != None and arg3 != None:
        sentenceDictionary, sentences, size = cleanText(arg1)
        window = int(arg3)
        numberofSentences = int(arg2)
        n = int(math.ceil(min(0.1 * size, 7 * math.log(size))))
        generatepositionaldistribution()
        keyphrases = textrank()
        t=summarize(arg1, keyphrases, numberofSentences)
        return(t)
    else:
        print("not enough parameters")

import streamlit as st
if __name__ == "__main__":
    #process(sys.argv[1])
                st.markdown("<h1    style='text-align:    center;'>Text
Summarization</h1>", unsafe_allow_html=True)
     uploaded_files= st.file_uploader('Upload text file', type=['txt'],
accept_multiple_files=False)
    if uploaded_files is not None:
     # To read file as bytes:
    #  bytes_data = uploaded_file.getvalue()
    #  st.write(bytes_data)
        bytes_data = uploaded_files.read().decode('utf-8')
        result=process(bytes_data)
        st.subheader("Input Text\n")
        st.markdown(
            f"<div style='text-align: justify;'>{bytes_data}</div>",
            unsafe_allow_html=True)
        st.subheader("Summarized text\n")
        st.markdown(
            f"<div style='text-align: justify;'>{result}</div>",
            unsafe_allow_html=True)
```

Sentence Extraction

For sentence extraction in Marathi, the goal is to rank entire sentences, and therefore a vertex is added to the graph for each sentence in the text. The co-occurrence which we have used earlier for keyword extraction cannot be applied here, since the text units in consideration are significantly larger than one or few words, and “co-occurrence” is not a meaningful relation for

such large contexts, it can give some meaningless data if text structure is not valid. Instead, we are defining a different relation, which determines a connection between two sentences if there is a “similarity” relation between them, where “similarity” is measured as a function of their content overlap. This relation between sentences is considered as process of “endorsement”: a sentence that addresses certain concepts in a text, gives the reader a “recommendation” to refer to other sentences in the text that address the same concepts, and therefore a link can be drawn between any two such sentences that share common content. The similarity of two sentences can be measured by the number of common tokens between two sentences, or it can be run through syntactic filters, which only count words of a certain syntactic category, e.g. all open class words, stopwords, etc. To avoid upholding long sentences, we are using a normalization factor, and break the content overlap.

Formally , given two sentences Si and Sj , with sentence being represented by the set of Ni words that appear in the sentence: Si = W1,W2,W3.....W Ni, the similarity of Si and Sj is Defined as:

$$\boldsymbol{Similarity(S_i, S_j) = \frac{\left|\{w_k | w_k \in S_i \& w_k \in S_j\}\right|}{\log(|S_i|) + \log(|S_j|)}}$$

The generated graph is extremely connected with each edge, with a weight associated indicating the strength of the connections established between various sentence pairs in the text.

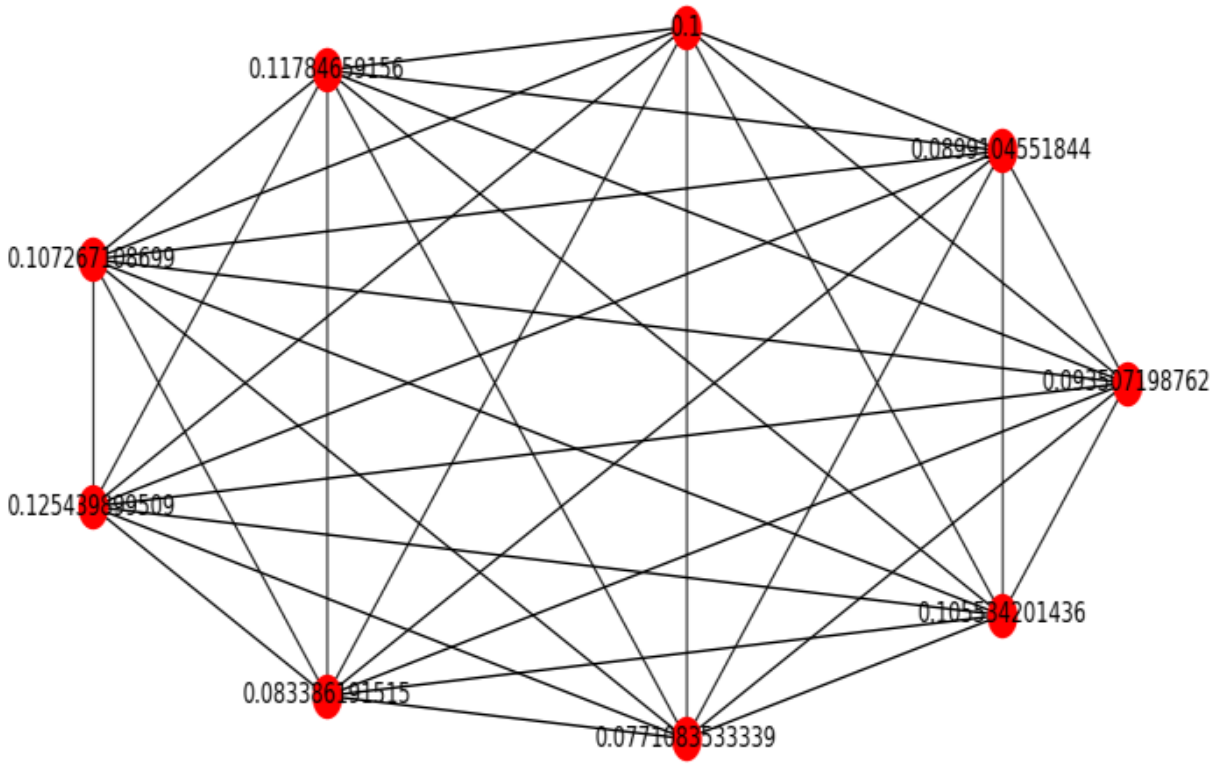


Fig 6.1.3:Sample graph build for sentence extraction

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from preprocess import cleanText
import networkx
import itertools
import math
import sys
import io
import json
import gradio as gr
sentenceDictionary = {};

def getSimilarity(sentenceID1, sentenceID2):
    commonWordCount = len(set(sentenceDictionary[sentenceID1]) &
set(sentenceDictionary[sentenceID2]))
    denominator = math.log(len(sentenceDictionary[sentenceID1])) +
math.log(len(sentenceDictionary[sentenceID2]))
    return commonWordCount/denominator if denominator else 0

def generateGraph(nodeList):
    graph = networkx.Graph()
```

```
    graph.add_nodes_from(nodeList)
    edgeList = list(itertools.product(nodeList, repeat=2))
    for edge in edgeList:
        graph.add_edge(edge[0], edge[1], weight=getSimilarity(edge[0],
edge[1]))
    return graph
def printDictionary():
    for key,val in sentenceDictionary.iteritems():
        print(str(key) + " : " + " ".join(sentenceDictionary[key])  )
def textRankSimilarity(filePath):
    global sentenceDictionary
    summarySentenceCount=5
    sentenceDictionary = {}
    sentences = []
    sentenceDictionary, sentences, size = cleanText(filePath)
    #printDictionary()
    graph = generateGraph(list(sentenceDictionary.keys()))
    # return(graph)
    pageRank = networkx.pagerank(graph)
    output = "\n".join([sentences[sentenceID] for sentenceID in
sorted(sorted(pageRank, key=pageRank.get,
reverse=True)[:summarySentenceCount])])
     return(output)

import streamlit as st
if __name__ == "__main__":
    st.markdown("<h1 style='text-align: center;'>Text
Summarization</h1>", unsafe_allow_html=True)
    uploaded_files= st.file_uploader('Upload text file', type=['txt'],
accept_multiple_files=False)
    if uploaded_files is not None:
        bytes_data = uploaded_files.read().decode('utf-8')
        result=textRankSimilarity(bytes_data)
        st.subheader("Input Text\n")
        st.markdown(f"<div style='text-align:
justify;'>{bytes_data}</div>",
            unsafe_allow_html=True)
        st.subheader("Summarized text\n")
        st.markdown(
            f"<div style='text-align: justify;'>{result}</div>",
            unsafe_allow_html=True)
    # textRankSimilarity(sys.argv[1])
```

Chapter 7

Conclusion

The main requirement of Text Summarization is keyword or sentence extraction and sentence ranking step. Extracting only the relevant keyphrases or sentences is important for the accuracy of the summarized text. Pre-processing is performed using the NLTK toolkit. Text Summarization in Marathi is obtained using Text Rank Algorithm. The Text Summarization system developed can be a part of various applications like text classification, question answering, legal texts summarization, news summarization, and headline generation.

The system currently summarizes text using extractive based summarization approach. The system has a lot of scope for further study with SOV (Subject Object Verb - Experimental) verification, TF-ISF (Term Frequency/ Inverse Sentence Frequency) to give more relevant and precise summary.

References

1. Vishal Gupta, G.S. Lehal, *"*A Survey of Text Mining Techniques and Applications*"*, *Journal of Emerging Technologies in Web Intelligence,* VOL. 1, NO. 1, 60- 76, AUGUST 2009

2. Rada Mihalcea and Paul Tarau, "TextRank: Bringing Order into Text". In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, 2004

3. Deepali Kailash Gaikwad. 2018. "Rule Based Text Summarization for Marathi Text ", *Journal of Global Research in Computer Science* 9, 5 (2018), 19–21

Acknowledgements

Our group would like to express our sincere thanks to **Director Bro. Jose Thuruthiyil**, our Principal **Dr. Sincy George** and the HOD of our department, **Dr. KavitaSonawane** for the precious platform to evolve and grow as a student.

We are highly indebted to our Teacher, Guide and Mentor **Mrs. Pradnya Sawant** for her constant guidance, knowledge, constructive feedback and motivation which have helped us achieve our objective for the report topic. We are thankful to the library staff of St. Francis Institute of Technology for their guidance and help to get the right information from the wide range of resources available.

We also want to thank our family members and friends for their patience and support throughout the project. We are also thankful to all the contributors of online forums, bloggers and YouTubers whose work and insights have helped us understand the perspectives of people around the globe.