

STUDENT INTERNSHIP PROGRAM (SIP) REPORT

# **ZnZ Systems**

**Submitted By**

Niranjan Patil

T184104.

0120160134.

### **SCHOOL OF COMPUTER ENGINEERING & TECHNOLOGY**

**MIT ACADEMY OF ENGINEERING ALANDI (D), PUNE**



CERTIFICATE

### This is to certify that the “Student Internship Program (SIP)” report submitted by **Niranjan Vinod Patil** PRN **0120160134** is work done by him and is submitted during **2018-2019** academic year.

**Faculty Mentor School-Internship Coordinator**

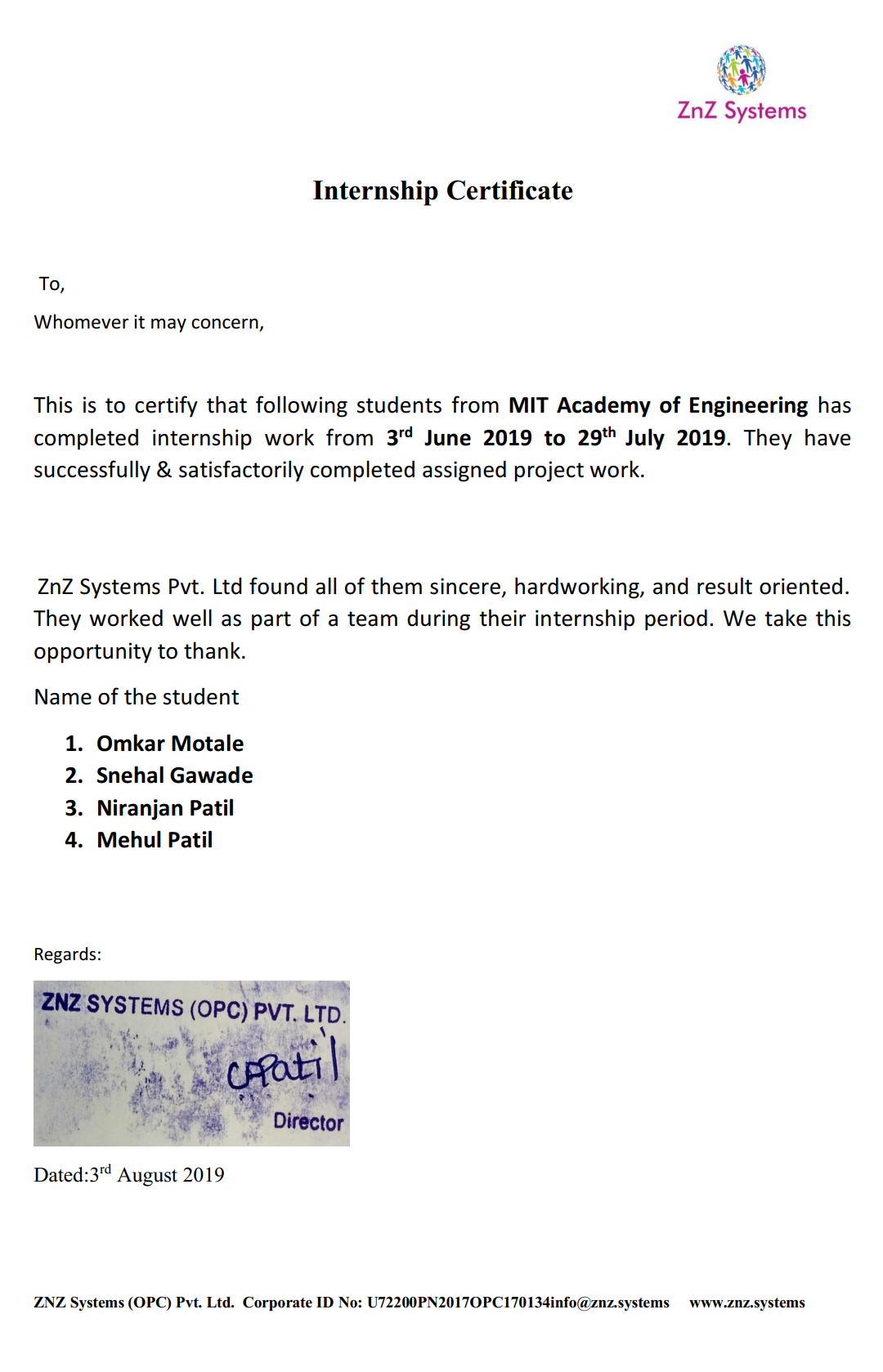
**(Mrs. Diptee Ghusse) (Mr. Rudragouda Patil)**

**Coordinator – SIP School Dean**

**(Mr. Senthil Kumar) (Prof. Ranjana Badre)**



**Internship certificate provided by the internship institution**





# **ACKNOWLEDGEMENT**

I, Mr. Niranjan Vinod Patil the student of School of Computer Engineering & Technology at MIT Academy of Engineering Pune (An Autonomous Institute), has been completed Summer Internship in “ZnZ Systems”.

I take this opportunity to record my profound gratitude and indebtedness to Mr. Chaitanya Patil, Director, ZnZ Systems for their inspiring guidance, valuable advices, constant encouragement and untiring supervision throughout my internship.

Finally, I would like to acknowledge and express my special thanks to Mr. Rudragouda Patil Assistant Professor, School of Computer Engineering & Technology, MIT Academy of Engineering Pune (An Autonomous Institute), my family and other team members for their patience, encouragement, support they have made during the period of this internship.

**(Niranjan Vinod Patil)**



## Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Topic** | | | **Page No.** |
| 1. | **Introduction** | | 06. |
|  | 1.1 | About the organization. | 06. |
|  | 1.2 | Scope of the work. | 07. |
| 2. | **Internship Discussion** | | 08. |
|  | 2.1 | Learning Experience. | 08. |
|  | 2.2 | Most challenging task performed. | 13. |
|  |  |  |  |
| 3. | **Conclusion** | | 30. |
| 4. | **Bibliography** | | 31. |



**1) Introduction.**

1.1 About the Organization.

ZnZ Systems (opc) Private Limited is a Private (One Person Company) incorporated on 26 April 2017. It is classified as Non-govt Company and is registered at Registrar of Companies, Pune. Its authorized share capital is Rs. 100,000 and its paid up capital is Rs. 100,000.

It is involved in Software publishing, consultancy and supply [Software publishing includes production, supply and documentation of ready-made (non-customized) software, operating systems software, business & other applications software, computer games software for all platforms. Consultancy includes providing the best solution in the form of custom software after analyzing the user’s needs and problems. Custom software also includes made-to-order software based on orders from specific users. Also, included are writing of software of any kind following directives of the users; software maintenance, web-page design]

Znz Systems (opc) Private Limited's Annual General Meeting (AGM) was last held on N/A and as per records from Ministry of Corporate Affairs (MCA), its balance sheet was last filed on 31 March 2018.  
Chaitanya Annasaheb Patil is director of Znz Systems (opc) Private Limited.  
Znz Systems (opc) Private Limited's Corporate Identification Number is (CIN) U72200PN2017OPC170134 and its registration number is 170134.Its Email address is 1134.chaitanya@gmail.com and its registered address is At Post Sankh, G N 1065, Tal Jath Sangli Sangli MH 416412 INDIA.



**The period of Internship: One and Half Month.**

**Starting Date:** 04 June 2019.

**Ending Date:** 14 July 2019.

**Time:** 10:00 AM – 5:00 PM.

1.2 Scope of the work.

**Purpose of Project:**

We all interact with an application which uses text summarization. Many of those applications are for the platform which publishes articles on daily news, entertainment, sports. With our busy schedule, we prefer to read the summary of that article before we decide to jump in for reading entire article. Reading a summary help us to identify the interest area, gives a brief context of the story. Summarization can be defined as a task of producing a concise and fluent summary while preserving key information and overall meaning.

**Objective of Project:**

Summarizing the contents of a text file.

**Functional Requirements:**

1. Accept the file as the input for summarizing.

2. Pass it through the code and obtain a summary of the contents of text file.

**2) Internship Discussion.**



2.1 Learning Experience.

1. Knowledge acquired:

**How text summarization works?**

In general there are two types of summarization, abstractive and extractive summarization.

1. **Abstractive Summarization:**

Abstractive methods select words based on semantic understanding; even those words did not appear in the source documents. It aims at producing important material in a new way. They interpret and examine the text using advanced natural language techniques in order to generate a new shorter text that conveys the most critical information from the original text.

It can be correlated to the way human reads a text article or blog post and then summarizes in their own word.

**Input document → understand context → semantics → create own summary.**

1. **Extractive Summarization:**

Extractive methods attempt to summarize articles by selecting a subset of words that retain the most important points. 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.**



**EXTRACTIVE SUMMARIZATION METHODS:**

A. Term Frequency-Inverse Document Frequency (TF-IDF) method

B. Cluster based method

C. Graph theoretic approach

D. Machine Learning approach

E. Text summarization with neural networks

F. Automatic text summarization based on fuzzy logic.

**Code flow:**

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

**Text Rank Algorithm:**

Before getting started with the Text Rank algorithm, there’s another algorithm which we should become familiar with – the PageRank algorithm. In fact, this actually inspired Text Rank! PageRank is used primarily for ranking web pages in online search results. Let’s quickly understand the basics of this algorithm with the help of an example.

**PageRank Algorithm**

Suppose we have 4 web pages — w1, w2, w3, and w4. These pages contain links pointing to one another. Some pages might have no link – these are called dangling pages.







* Web page w1 has links directing to w2 and w4
* w2 has links for w3 and w1
* w4 has links only for the web page w1
* w3 has no links and hence it will be called a dangling page

In order to rank these pages, we would have to compute a score called the **PageRank score**. This score is the probability of a user visiting that page.

To capture the probabilities of users navigating from one page to another, we will create a square **matrix M**, having n rows and n columns, where **n** is the number of web pages.





Each element of this matrix denotes the probability of a user transitioning from one web page to another. For example, the highlighted cell below contains the probability of transition from w1 to w2.



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

1. 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)**

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

3. 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)**

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





Finally, the values in this matrix will be updated in an iterative fashion to arrive at the web page rankings.

Let’s understand the TextRank algorithm, now that we have a grasp on PageRank. I have listed the similarities between these two algorithms below:

* In place of web pages, we use sentences
* Similarity between any two sentences is used as an equivalent to the web page transition probability
* The similarity scores are stored in a square matrix, similar to the matrix M used for PageRank.

**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. TextRank Algorithm Procedure**



The first step would be to concatenate all the text contained in the articles

* Then split the text into individual sentences
* In the next step, we will find vector representation (word embeddings) for each and every sentence
* Similarities between sentence vectors are then calculated and stored in a matrix
* The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation
* Finally, a certain number of top-ranked sentences form the final summary.
  1. Most Challenging task performed.

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.

For Abstractive Text summarization we will be building a text summarization graphical user interface (GUI) app using tkinter Method.

**Summaryzer\_GUI:** Simply text/ document summarization GUI with python tkinter Method and features of GUI are as follow:

1. **Input Area:** Enter text which has to be summarized.
2. **Output Area:** Display summary of given text after Choosing summarize option.



1. **Reset:** Clear / Empty Input Area.
2. **Clear Result:** Clear / Empty Output Area.
3. **Save:** Save text of output area in text file format.
4. **Fetch text from URL and summarize it.**
5. **Summarize files as well as links.**
6. **Compare various summarizers using Spacy, Sumy, Gensim and NLTK.**

**Prerequisites:**

1. Python 3
2. Spacy
3. NLTK
4. Gensim
5. Sumy

**Python GUI – tkinter**

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.  
**To create a tkinter:**

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.



**Summary of text using NLTK:**

1. Remove stop words for the analysis
2. Create frequency table of words - how many times each word appears in the text
3. Assign score to each sentence depending on the words it contains and the frequency table
4. Build summary by adding every sentence above a certain score threshold.

### **What are stop words?**

Any word that does not add a value to the meaning of a sentence.  
For example, let's say we have the sentence is “A group of people run every day from a bank in Alafaya to the nearest Chipotle”

By removing the sentences stop words, we can narrow the number of words and preserve the meaning: “Group of people run every day from bank Alafaya to nearest Chipotl”

We usually remove stop words from the analyzed text as knowing their frequency doesn't give any insight to the body of text. In this example, we removed the instances of the words a, in, and the.

There are two NLTK libraries that will be necessary for building an efficient summarizer.

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize



### **Corpus**

Corpus means a collection of text. It could be data sets of poems by a certain poet, bodies of work by a certain author, etc. In this case, we are going to use a data set of pre-determined stop words.

### **2. Tokenizers**

Basically, it divides a text into a series of tokens. There are three main tokenizers - word, sentence, and regex tokenizer. For this specific project, we will only use the word and sentence tokenizer.

## Removing stop words and making frequency table

First, we create two arrays - one for stop words, and one for every word in the body of text.

Let's use text as the original body of text.

stopWords = set(stopwords.words("english"))

word\_tokenize(raw\_text)

Second, we create a dictionary for the word frequency table. For this, we should only use the words that are not part of the stopWords array.

word\_frequencies = {}

for word in nltk.word\_tokenize(raw\_text):

if word not in stopWords:

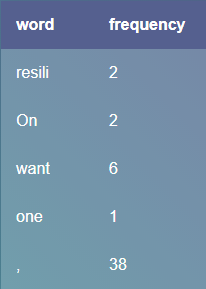
if word not in word\_frequencies.keys():

word\_frequencies[word] = 1

else:

word\_frequencies[word] += 1





Now, we can use the freqTable dictionary over every sentence to know which sentences have the most relevant insight to the overall purpose of the text.

## Assigning a score to every sentence

We already have a sentence tokenizer, so we just need to run the sent\_tokenize() method to create the array of sentences. Secondly, we will need a dictionary to keep the score of each sentence, this way we can later go through the dictionary to generate the summary.

maximum\_frequncy = max(word\_frequencies.values())

for word in word\_frequencies.keys():

word\_frequencies[word]= (word\_frequencies[word]/maximum\_frequncy)

sentence\_list = nltk.sent\_tokenize(raw\_text)

sentence\_scores = {}



Now it's time to go through every sentence and give it a score depending on the words it has. There are many algorithms to do this - basically, any consistent way to score a sentence by its words will work. I went for a basic algorithm: adding the frequency of every non-stop word in a sentence.

for sent in sentence\_list:

for word in nltk.word\_tokenize(sent.lower()):

if word in word\_frequencies.keys():

if len(sent.split(' ')) < 30:

if sent not in sentence\_scores.keys():

sentence\_scores[sent] = word\_frequencies[word]

else:

sentence\_scores[sent] += word\_frequencies[word]

### So, what value can we use to compare our scores to?

A simple approach to this question is to find the average score of a sentence.

**Summary of text using Spacy:**

Basic idea for creating a summary of any document includes the following:

1. Text Preprocessing (remove stop words, punctuation).
2. Frequency table of words/Word Frequency Distribution – how many times each word appears in the document
3. Score each sentence depending on the words it contains and the frequency table
4. Build summary by joining every sentence above a certain score limit.



# NLP Pkgs

import spacy # Load Packages

nlp = spacy.load('en')

# Pkgs for Normalizing Text

from spacy.lang.en.stop\_words import STOP\_WORDS

from string import punctuation

# Import Heapq for Finding the Top N Sentences

from heapq import nlargest

def text\_summarizer(raw\_docx):

raw\_text = raw\_docx

docx = nlp(raw\_text)

stopwords = list(STOP\_WORDS)

# Build Word Frequency # word.text is tokenization in spacy

#### Word Frequency Table

1. Dictionary Of Words And Their Counts
2. How Many Times Each Word Appears In The Document
3. Using Non-Stopwords

word\_frequencies = {}

for word in docx:

if word.text not in stopwords:

if word.text not in word\_frequencies.keys():

word\_frequencies[word.text] = 1

else:

word\_frequencies[word.text] += 1

#### Maximum Word Frequency

1. Find The Weighted Frequency
2. Each Word Over Most Occurring Word
3. Long Sentence Over Short Sentence



maximum\_frequncy = max(word\_frequencies.values())

for word in word\_frequencies.keys():

word\_frequencies[word] = (word\_frequencies[word]/maximum\_frequncy)

#### Sentence Score And Ranking Of Words In Each Sentence

1. Sentence Tokens
2. Scoring Every Sentence Based On Number Of Words
3. Non Stopwords In Our Word Frequency Table

# Sentence Tokens

sentence\_list = [ sentence for sentence in docx.sents ]

# Sentence Scores via comparing each word with sentence

sentence\_scores = {}

for sent in sentence\_list:

for word in sent:

if word.text.lower() in word\_frequencies.keys():

if len(sent.text.split(' ')) < 30:

if sent not in sentence\_scores.keys():

sentence\_scores[sent] = word\_frequencies[word.text.lower()]

else:

sentence\_scores[sent] += word\_frequencies[word.text.lower()]

summarized\_sentences = nlargest(7, sentence\_scores, key=sentence\_scores.get)

final\_sentences = [ w.text for w in summarized\_sentences ]

summary = ' '.join(final\_sentences)

return summary

**Summary of text using Gensim**

**Gensim** is a free Python library designed to automatically extract semantic topics from documents. Summarizing is based on ranks of text sentences using a variation of the TextRank algorithm.



# Package for gensim

From gensim.summarization.summarizer import summarize

Print(summarize(text))

**TextRank**is a general purpose, **graph based** ranking algorithm for NLP. TextRank is an automatic summarisation technique. Graph-based ranking algorithms are a way for deciding the importance of a vertex within a graph, based on global information recursively drawn from the entire graph.

**TextRank Model:**

The basic idea implemented by a graph-based ranking model is that ofvoting *or*recommendation. When one vertex links to another one, it is basically casting a vote for that vertex. The higher the number of votes cast for a vertex, the higher the importance of that vertex.

**Text as a graph -**

We have to build a graph that represents the text, interconnects words or other text entities with meaningful relations.

TextRank includes two NLP tasks-

1. Keyword extraction task
2. Sentence extraction task

**Keyword Extraction -**

The task of keyword extraction algorithm is to automatically identify in a text a set of terms that best describe the document.

The simplest possible approach is to use a frequency criterion.

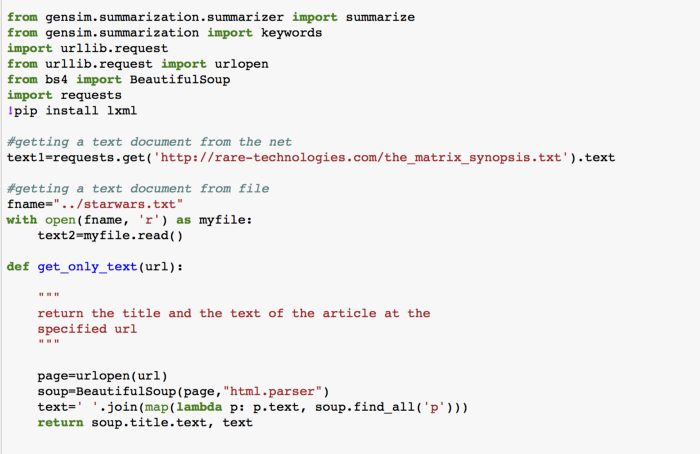
HOWEVER, this leads to poor results. The TextRank keyword extraction algorithm is fully **unsupervised. No training is necessary.**



**Sentence Extraction -**

TextRank is very well suited for applications involving entire sentences, since it allows for a ranking over text units that is recursively computed based on information drawn from the entire text.

To apply TextRank, we first build a graph associated with the text, where the graph vertices are representative for the units to be ranked. The goal is to rank entire sentences, therefore, a vertex is added to the graph for each sentence in the text.







**PageRank Algorithm -**

It is the foundation of TextRank.

* PageRank used by Google search.
* Used to compute the rank of web pages. It is not named after its use (ranking pages) but after its creator Larry Page.

**Fundamentals -**

* Important pages are linked by important pages.
* The PageRank value of a page is the probability of a user visiting that page.

In TextRank, the only difference is that we consider sentences instead of pages.



**Summary of text using Sumy:**

**How To Installation**

pip install sumy

Sumy offers several algorithms and methods for summarization such as:

1. Luhn – heurestic method
2. Latent Semantic Analysis
3. Edmundson heurestic method with previous statistic research
4. LexRank – Unsupervised approach inspired by algorithms PageRank and HITS
5. TextRank
6. SumBasic – Method that is often used as a baseline in the literature
7. KL-Sum – Method that greedily adds sentences to a summary so long as it decreases the KL Divergence.

# Load Packages

import sumy

from sumy.parsers.plaintext import PlaintextParser

from sumy.nlp.tokenizers import Tokenizer

from sumy.summarizers.lex\_rank import LexRankSummarizer

document1 = ScrolledText(tab2, height = 10)

# For Strings

parser = PlaintextParser.from\_string(document1,Tokenizer("english"))

# For Files

parser = PlaintextParser.from\_file(file, Tokenizer("english"))



**USING LEXRANK**

Unsupervised approach to text summarization based on graph-based centrality scoring of sentences. The main idea is that sentences “recommend” other similar sentences to the reader. Thus, if one sentence is very similar to many others, it will likely be a sentence of great importance

Standalone pkg pip install lexrank

# Using LexRank

summarizer = LexRankSummarizer()

#Summarize the document with 2 sentences

summary = summarizer(parser.document, 2)

for sentence in summary:

print(sentence)

**USING LUHN**

Based on frequency of most important words

from sumy.summarizers.luhn import LuhnSummarizer

summarizer\_luhn = LuhnSummarizer()

summary\_1 =summarizer\_luhn(parser.document,2)

for sentence in summary\_1:

print(sentence)



**USING LSA**

Based on term frequency techniques with singular value decomposition to summarize texts.

from sumy.summarizers.lsa import LsaSummarizer

summarizer\_lsa = LsaSummarizer()

summary\_2 =summarizer\_lsa(parser.document,2)

for sentence in summary\_2:

print(sentence)

## Alternative Method using stopwords

from sumy.nlp.stemmers import Stemmer

from sumy.utils import get\_stop\_words

summarizer\_lsa2 = LsaSummarizer()

summarizer\_lsa2 = LsaSummarizer(Stemmer("english"))

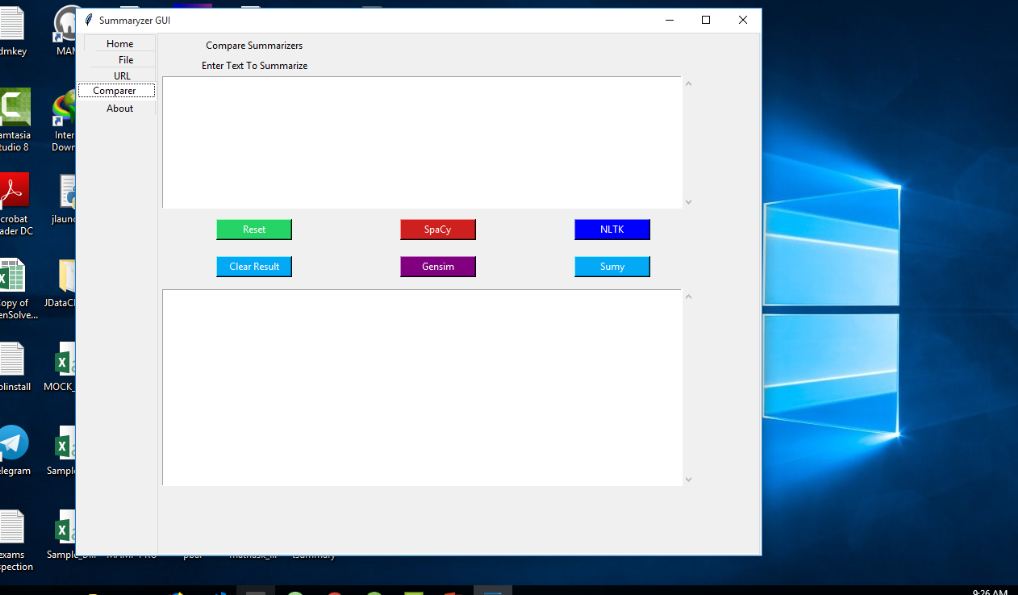
summarizer\_lsa2.stop\_words = get\_stop\_words("english")

for sentence in summarizer\_lsa2(parser.document,2):

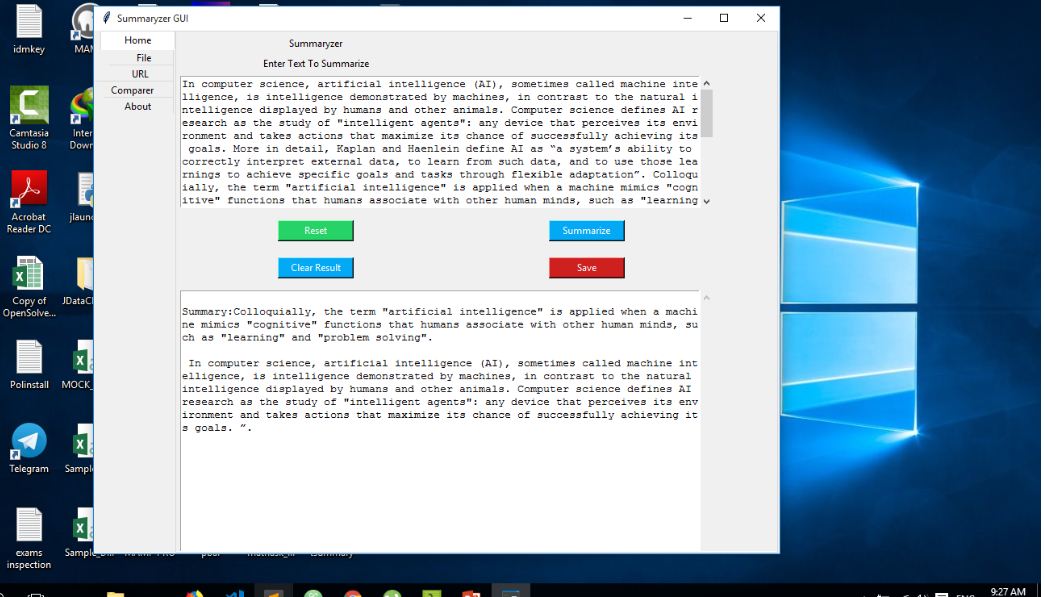
print(sentence)



**Screenshots of GUI:**

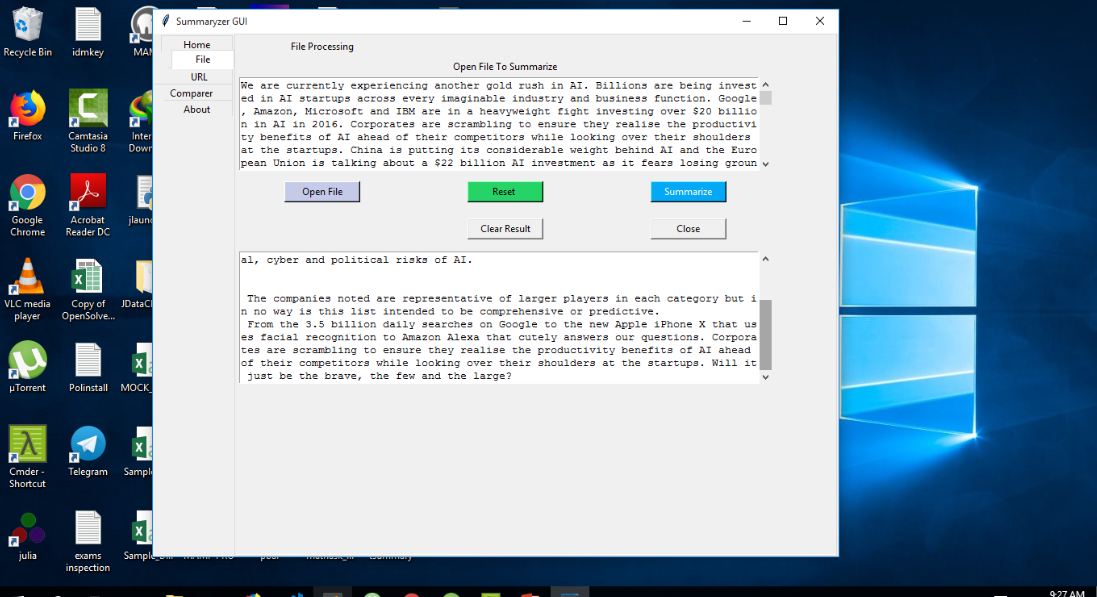
****

**Fig 3. GUI Home Page**

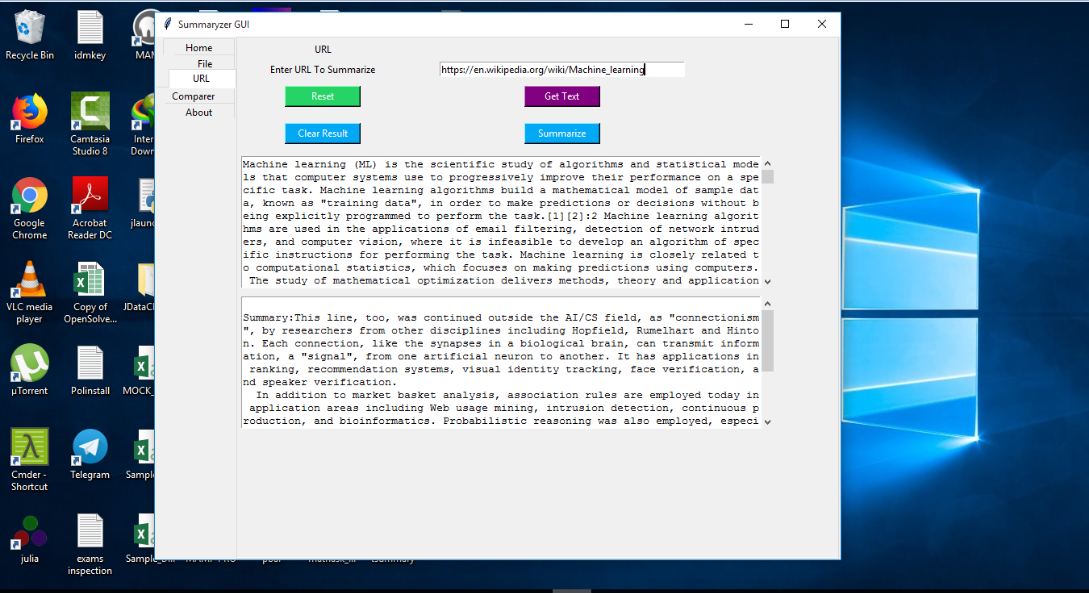
****

**Fig 4. GUI Home Page when text is to be summarized**



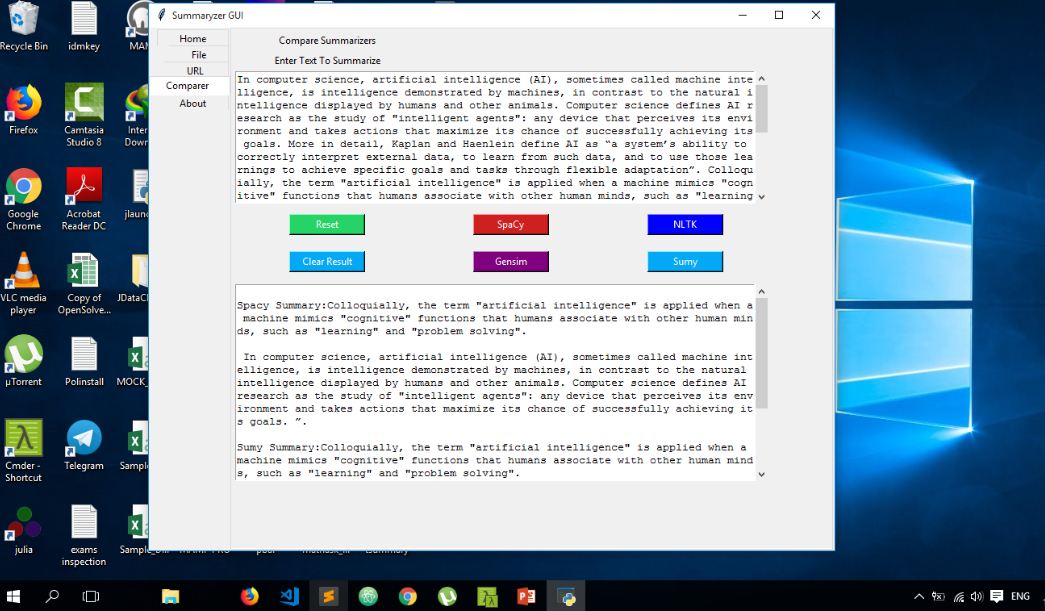


**Fig 5. GUI File Page**



**Fig 6. GUI URL Page**





**Fig 7. GUI Comparer Page**

**3) Conclusion.**

Automatic text summarization is an old challenge but the current research direction diverts towards emerging trends in biomedicine, product review, education domains, emails and blogs. This is due to the fact that there is information overload in these areas, especially on the World Wide Web. Automated summarization is an important area in NLP (Natural Language Processing) research. It consists of automatically creating a summary of one or more texts. The purpose of extractive document summarization is to automatically select a number of indicative sentences, passages, or paragraphs from the original document .Text summarization approaches based on Neural Network, Graph Theoretic, Fuzzy and Cluster have, to an extent, succeeded in making an effective summary of a document. Both extractive and abstractive methods have been researched. Most summarization techniques are based on extractive methods. Abstractive method is similar to summaries made by humans. Abstractive summarization as of now requires heavy machinery for language generation and is difficult to replicate into the domain specific areas.

**Future Scope:**



Cross-language text summarization (source in some language and summary in another language)



**4) Bibliography.**

1. [ZnZ System Website.](Final%20Report.docx)
2. [https://github.com/Jcharis/NLP-Web-Apps](Final%20Report.docx)
3. [https://medium.com/@shivangisareen/text-summarisation-with-gensim-textrank-46bbb3401289](Final%20Report.docx)
4. [https://towardsdatascience.com/text-summarization-in-python-76c0a41f0dc4](Final%20Report.docx)
5. [https://jcharistech.wordpress.com/2019/01/05/how-to-summarize-text-or-document-with-sumy/](Final%20Report.docx)
6. [https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/](Final%20Report.docx)
7. [https://youtu.be/KKpf0EcgkUU](Final%20Report.docx)