Extractive Summarization and Simplification of Scholarly Literature

By

Nilav Bharatkumar Vaghasiya

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington

In Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

Supervisor: Dr. Elizabeth Diaz

May 2020

Acknowledgement

It is my privilege and duty to acknowledge the kind of help and guidance received from several people in the completion of my master's Thesis at The University of Texas at Arlington. It would not have been possible to complete this project without their valuable help, cooperation and guidance.

I would like to extend my sincere gratitude to my mentor and supervisor Dr. Elizabeth Diaz, Senior Lecturer, Department of Computer Science and Engineering, The University of Texas at Arlington, for her continuous guidance, help, support, motivation and constructive suggestions throughout the period of my thesis. This thesis would not have been possible without her constant encouragement and support.

I would like to thank the thesis committee members, Dr. Hao Che, Professor, Department of Computer Science and Engineering, The University of Texas at Arlington and Dr. David Levine, Distinguished Senior Lecturer, Department of Computer Science and Engineering, The University of Texas at Arlington for their invaluable cooperation, guidance and support.

And last but not the least, I would like to thank my parents and friends, without whose constant help, motivation and support, this thesis would not have been possible.

Abstract

EXTRACTIVE SUMMARIZATION
AND SIMPLIFICATION OF SCHOLARLY LITERATURE

Nilav Bharatkumar Vaghasiya, MS

The University of Texas at Arlington, 2020

Supervising Professor: Dr. Elizabeth Diaz

Research papers and journals have always played a crucial role in the field of research and development. However, these research papers usually have a complex usage of language which limits the range of target readers. The language and the terms used in these literary works can make the concept or the topic tough to understand for a naive reader. The goal of this project is to simplify a complex piece of literature into something meaningful without sounding verbose.

The idea is based upon Nobel Prize-winning physicist Richard Feynman's learning technique known as the Feynman Technique that emphasizes the usage of words that are simple to understand. Such simplification can help a researcher gain more information from more literature in less amount of time. The proposed system can also be used to simplify other text documents apart from research papers and journals. This novel model based on simplicity can be considered a new learning model.

Table of Contents

List of Illustrations

List of Table

Chapter 1

Introduction

Simplification of text involves summarization using different approaches like abstractive summarization and extractive summarization. For this system we are considering extractive summarization. Extractive Summarization means identifying critical sections of the literature followed by generating word-by-word subset of sentences from the original text. This is different from abstractive summarization, where it reproduces important information after interpreting and examining the test using different language processing techniques to give a shorter text that translates the most important information interpreted from the original one.

The system first summarizes the text and simplifies it to a level at which a naive user can gain insights from the summary. Simplification of complex literature makes it easier to interpret and understand the context. The process thereby reduces reading time. It makes the selection process easier for research papers and helps in indexing. Also, machine generated summaries are unbiased unlike human generated summaries.

We are proposing a novel model that has an input, '*n*' files of type PDF with a uniform structure, aiming to summarize then simplify it into one document. The system first identifies the structure of the papers and thereby identifies text from the critical sections. Once the text is extracted, it is then subjected to tokenization, weighing of tokens based on their relevance and then constructing the summary. This summary is further subjected to a simplification system where complex words are further broken down into simpler terms. This process is performed for '*n*' number of papers and the summaries are then presented as a single consolidated document.

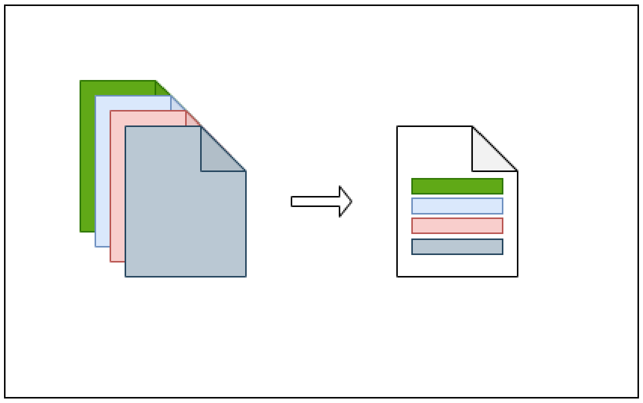


Fig 1.1 Summarizing 'n' documents into a single document

The proposed system is implemented as a web application and uses four different techniques to summarize the text and two different techniques to simplify a given text.

The paper discusses previous works, system design, architecture, implementation, observations and analysis and future work for the mentioned system.

Chapter 2

Motivation

"If you can't explain it simply, you don't understand it well enough." is a very famous quote by Richard Feynman [1]. He was an American Theoretical Physicist, well known for his learning technique. He could simplify almost any complex topic into something simple without any loss in the brevity of the topic. The learning technique involved four crucial steps:

- Write down the complex concept on a piece of paper: One should write down the complex topic on the piece of paper that is to be learned. Writing it down is important as it helps increase the retention.
- Try to elaborate the concept using simple language: Try to do this by simplifying the concept to the very basics. Try not to use complex subject-specific words for better understanding.
- Identify complex subtopics: Find the tough subtopics that you don't know. If need be, review some literature for the subtopic and read about it until its thoroughly understood.
- Replace complex words with simple terms: If you have understood the topic and have no doubts regarding any subtopic, you should be able to replace any complex and subject specific terms with simpler words.

On noticing above steps, we can conclude that simplification is the key to learning a new concept. As simple as the steps seems, it comprises of two core principles. The first principle of his technique revolved around simplicity of a subject, which means no use of complex jargon. The second principle revolved around brevity or briefness. These two principles can be used to simplify any kind of complex subject text.

There is no single user-friendly system or proof of this concept which would summarize multiple research documents at once with a simple jargon and thereby help a user to learn more about these subjects in a lesser span of time.

Many summarization systems, both abstractive and extractive have existed for a long time. The automatic creation of literature abstracts was published by H.P Luhn in 1958 [2]. Similarly, summarization using TextRank by Federico Barrios, Federico Lopez, Luis Argerich and Rosa Wachenchauzer [3] was also implemented as gensim summarization. These are few examples out of many summarization models and systems. These systems demonstrated their performance and were compared to the previous systems. Only a few systems like SummIt: A Tool for Extractive Summarization, Discovery, and Analysis by IBM [4] served an actual purpose of summarizing.

The task of studying a subject through research papers involves surveying a huge amount of literature. Researching about a subject requires extensive literature survey. This includes looking up for different research articles, documents and their related literature as well. A single research article or a document has at least 5 citations from different research articles and it takes about 30 minutes to go through just the research article without browsing its reference literature. This reading time gets multiplied into the number of reference papers cited in the article. The reading time can be decreased if the cited papers and the base paper can be surveyed together in a simple and concise manner to get a fair idea about the subject. This idea led to the motivation to develop a user-friendly summarization and simplification system for text articles. The motivation was to create a tool for researchers to be able to understand a subject using the system. The key idea is to use Feynman technique and learn a topic with conjunction to using the system to effectively read through the literature required to understand the concept. The system has a wide variety of applications apart from summarizing scholarly articles.

The system can be used to summarize and simplify web articles and provide a gist to the user without them having to go through the article. The system can automatically reduce the dimensionality of research literature and thereby making the process of presenting and explaining a concept much easier.

Chapter 3
Previous Works

Most of the previous work on text summarization has been based on known corpus. The supervised approach for extractive summarization of scientific literature by ED Collins [5] explored a new dataset for summarization by exploiting an existing resource - Science Direct, where many journals required the authors to submit highlight statements along with their articles. This is different from the abstract of the article. These highlights could further be used as ‘gold’ statements for the summarization system. The author even introduces HighlightROUGE for this specific dataset having highlights as ‘gold’ statements. Other metrics like AbstractROUGE, a metric presented in the mentioned work is simply the ROUGE-L score of the sentence and the abstract. The system discussed in the above-mentioned work has a combination of 4 possible inputs: Sentence encoded with an RNN, a vector representation of the abstract of the said paper, one of the 8 features and sentence representation by average of every non-stop word vector in sentence. Multiple ways to implement the system are demonstrated. It showcases how their metric AbstractROUGE increases summarization performance. Even though the system does a great job at summarizing scientific articles, the requirement for domain-based knowledge is still present. There is no user-friendly system for using the system.[5]

Projects like ScisummNet [6] have also facilitated research in ‘citation-aware’ scientific paper summarization. It is useful for sequence2sequence summarization. Another work by Wen Xiao and Giuseppe Carenini [7] shows distributed representation of both the global (the whole document) and the local context (e.g., the section/topic) when deciding if a sentence should be included in the summary. Their inspiration for work was natural topic-oriented structure of human-written long documents.

When we observe all the previous summarizing systems, it is evident that a natural language processing pipeline helps obtain the structure of the input text, followed by evaluation of the corpus based on some rank or significance factor, after which the extract is given as output. There are various summarizing systems available for use but most of them either require a dataset or have not been developed keeping naïve users in mind. There is a need for a system which does not require domain knowledge and can work for multiple documents, irrespective of their background domain.

Chapter 4

System design and architecture

The proposed system is a web application, which runs on a flask server. It takes document files as input and stores it in a temporary volatile directory. At the server side, most of the functionalities are handled by Python, which also executes commands to execute intermediate Jar files by using Java Runtime Environment. The web application is hosted on a local server with debug mode on.

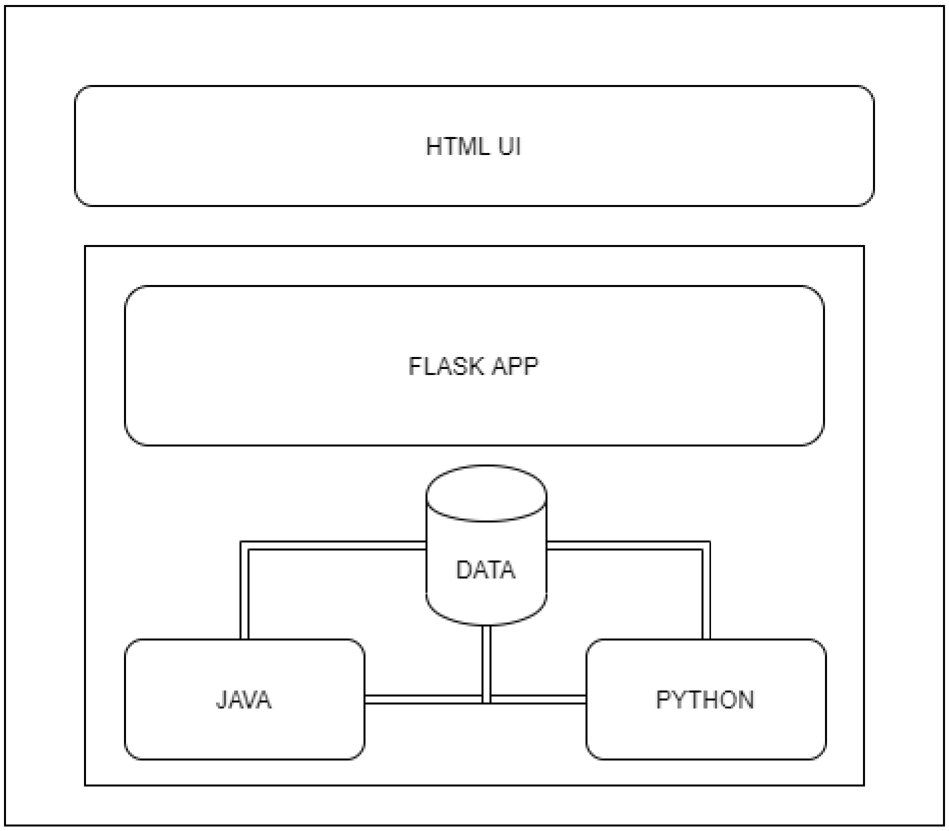


Fig. 4.1 System Architecture

The above figure shows the system architecture and the key elements of the system. The input for the web application is supposed to be files having a portable document file format i.e. PDF. This is the final output is a consolidated document having relevant information about the summarized and simplified paper along with the summary and the method used for the summary generation.

We choose flask as the framework to maximize the capabilities of python and help aggregate all the singular functionalities. The system was developed using python 3.6.0 and Java 14 was used to compile two packages which are then used with a command line interface. Initially the system was conceived as a group of scripts and the input was in the form of raw text. This architecture was not user friendly and needed a lot of manual work for handling the input and the output for different modules. This led to develop the application as a web interface. However, each of the underlying modules are almost independent and can be used as individual modules.

The detailed architecture for the above system comprises of six modules. Each module takes in an input and gives an output and performs forward propagation. Since the output of these modules are input for other modules, we implement serialization to avoid any exceptions while execution.

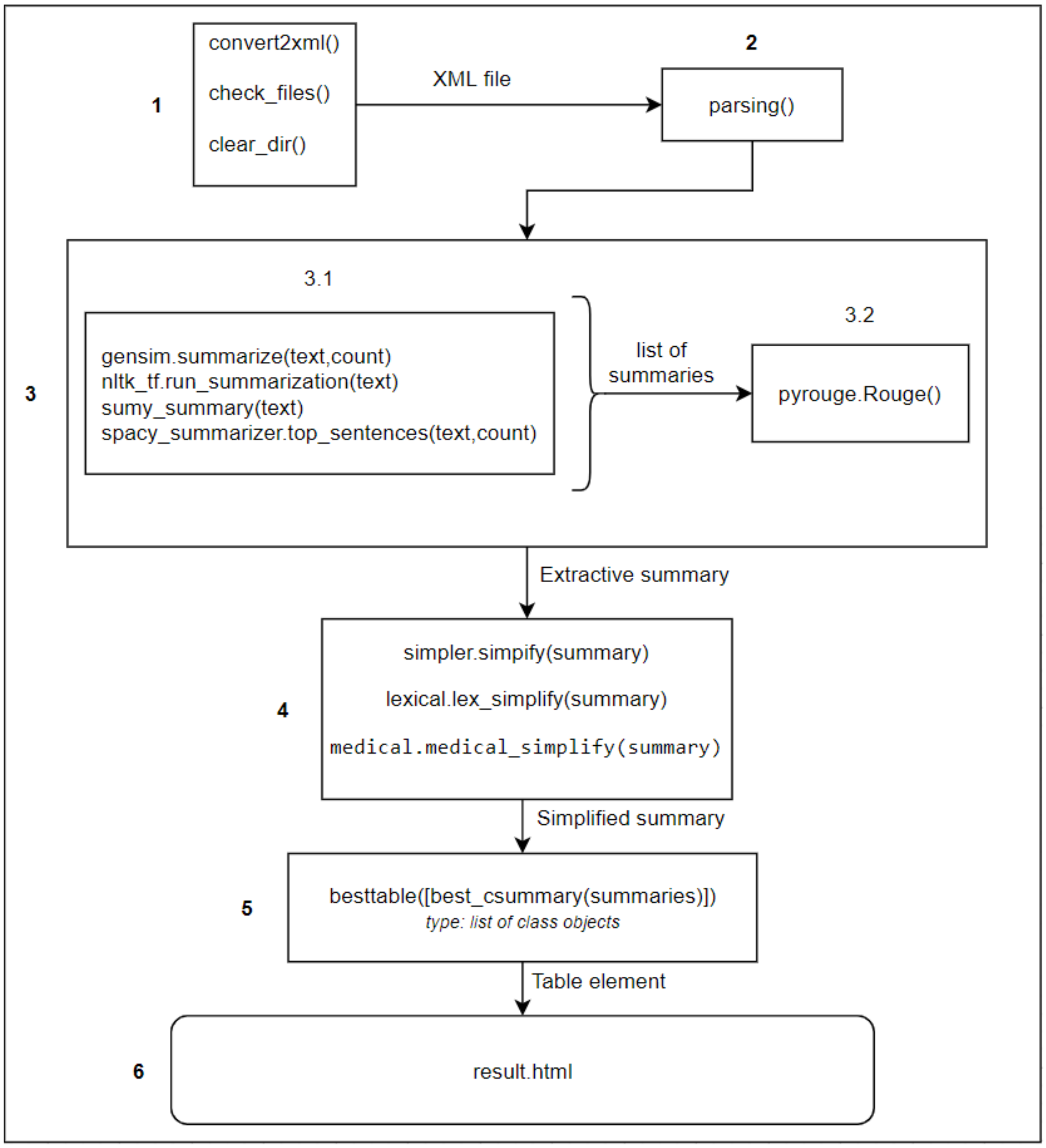

1
convert2xml()
check_files()
clear_dir()
XML file
2
parsing()
3
3.1
gensim.summarize(text,count)
nltk_tf.run_summarization(text)
sumy_summary(text)
spacy_summarizer.top_sentences(text,count)
list of summaries
3.2
pyrouge.Rouge()
Extractive summary
4
simpler.simpify(summary)
lexical.lex_simplify(summary)
medical.medical_simplify(summary)
Simplified summary
5
besttable([best_csummary(summaries)])
type: list of class objects
Table element
6
result.html

Fig 4.2 Workflow and detailed architecture of the system

The modules shown in the above figure as discussed below for detailed understanding:

1) Once the data is stored it is processed by *cfd.py* module. It has three functions: *convert2xml( ), check_files( ), clear_dir( ).* The *convert2xml( )* function searches for all the PDF documents in the directory and converts them to xml by using the CERMINE [8] library . The function executes terminal commands for the jar file and thereby initiating the conversion using the java library. The *check_files( )* function traverses the data directory and appends the path of the generated XML files into a list and returns. The *clear_dir( )* function is used to clear the generated xml files after they are processed. This helps to decrease the number of files in the directory and therefore decreasing traverse time for *check_files( )* . The output is an XMLfile .

2) The XML file is now passed on to the *xml_parser.py* which has *parsing( )* function. This function takes a list of paths as an input. It opens up the XML files and uses *xml.etree.ElementTree* to traverse them and extract information. The output for this function is a dictionary of dictionaries containing the textual information about the document. Since it takes a list as an input, it can handle multiple XML files at once.

3) This module is responsible for extractive summarization and evaluation and hence it is divided into two sub-modules. The module marked by 3.1 in the figure is responsible for all different extractive summarization techniques. There is no explicit module in the system which aggregates all the four summarization functions into one. There are four functions implementing four different ways to

summarize text. These functions are discussed in detail in upcoming chapters. The module defined in 3.2 is used for evaluating the summary metric scores and selecting the best generated summary.

4) Once the summary is selected for a document it is then passed on to the simplification methods. If the summary is to be subjected to context analysis, *simpler.simplify(summary)* is called. If the summary is to be subjected to lexical simplification using synonyms, *lexical.lex_simplify(summary)* is called. Else if the summary is to be subjected to 'medical science' subject based dictionary simplification, *medical.medical_simplify(summary)* is called. The option to choose one of these functions is given in the form of radio buttons while taking input documents from the user.

5) The simplified summary along with the document information stored in the form of a dictionary is now ready to be displayed. The dictionary items are initialized in class *best_csummary( )*. This creates an instance of the class which is then appended to a list. Once all the documents from the dictionary are appended as objects of class *best_csummary( ),* the list is then subjected to *besttable( )* and there by converted to an HTML table element using *flask_table* package. Different format of the table requires different classes.

6) Once the table element is passed as a parameter inside the *render_template( )¸*the result.html template renders the table element generated and thereby prints the table in the form of html. The table element can be formatted as per the need.

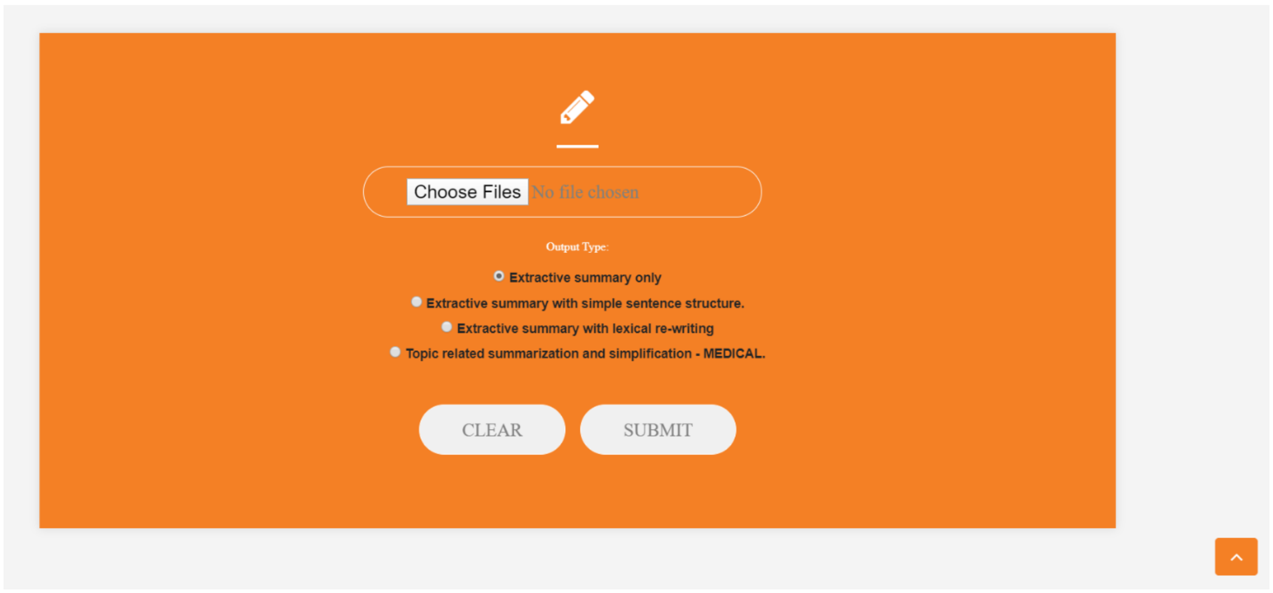
Chapter 5

Implementation

The implementation for the system is divided into following parts:

- Data Aggregation
- Information Extraction
- Processing
- Summarization and Evaluation
  - Weighted term frequency method using *Spacy* [9]
  - Latent Semantic Analysis [13] based summarization using *Sumy* [10]
  - Textrank [14] based summarization using *Gensim* [11]
  - TF-IDF [15] based summarization using *NLTK* [12]
- Simplification
  - Context Breakdown
  - Term Re-writing

5.1 Data Aggregation: For the ease of use of the proposed system, the user interface is developed in HTML. The frontend facilitates the user to upload the batch of documents in the PDF format. The uploaded documents are stored in a temporary directory on the local host for the time period of the session. The data, once summarized, is deleted from the host system therefore no privacy issues are to be tackled. The input is taken as form values with two main parameters: *'n'* number of structured documents and an option to simplify the summary with the selection of corresponding radio button.



Output Type:

[✓ Extractive summary only]
Extractive summary with simple sentence structure.
Extractive summary with lexical re-writing
Topic related summarization and simplification - MEDICAL.

Fig 5.1 HTML user interface for data aggregation

The input files need to be a literature of scholarly nature - it should at least have a title, an abstract and literature to be summarized. The system handles most of the renowned research publication's document format but may fail if the structure of the document is unconventional.

5.2 Information Extraction: Once the document files have been aggregated and stored in the local host directory, we subject these files to a java archive compiled from the source code of a library called CERMINE. CERMINE is an abbreviation for Content Extractor and Miner. It is an open source java library developed by Interdisciplinary Centre for Mathematical and Computational Modelling (ICM). CERMINE is used to extract the title of the input document, information about the journal, keywords, abstract, names of the author and all possible text information present in the document.

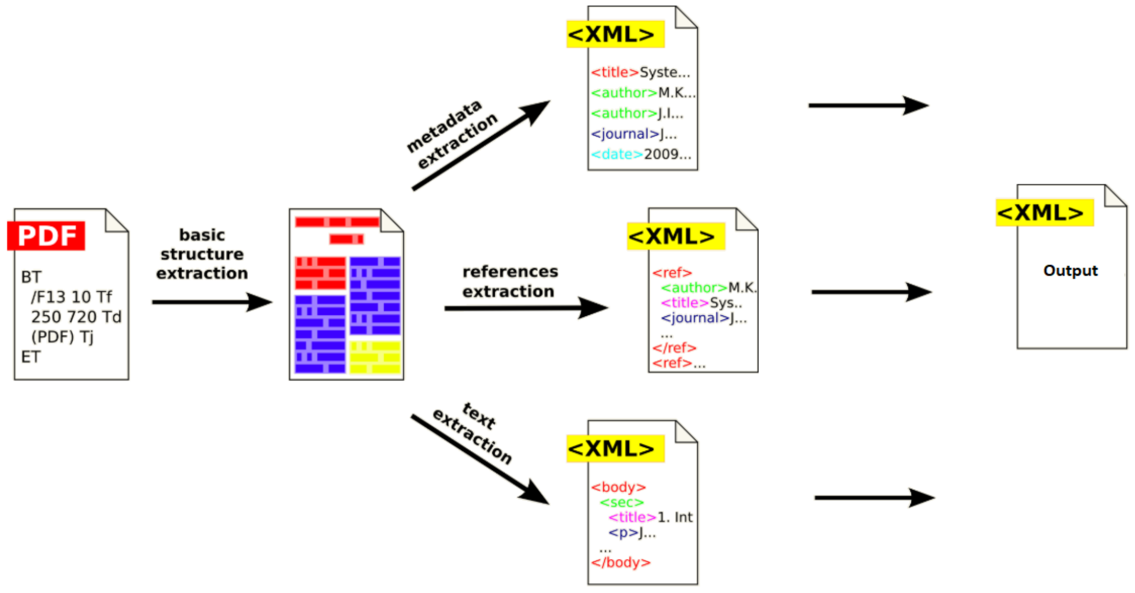


Fig 5.2 CERMINE Architecture

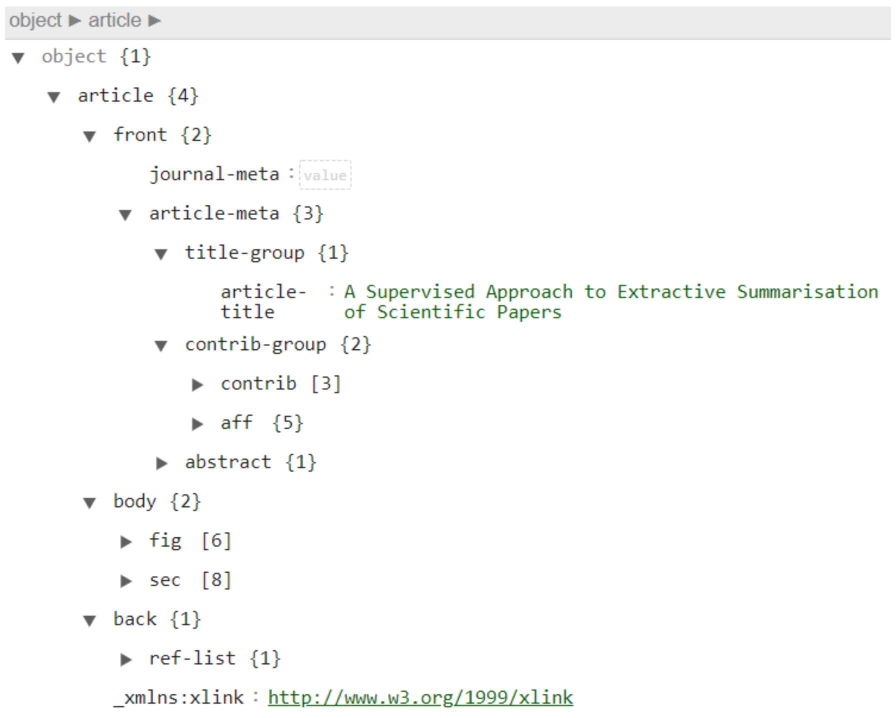
Extraction of the basic document skeleton from the pdf produces a chronological and hierarchical structure representing the document. This structure includes pages, lines, zones, words, and characters. The order for reading the document is determined with zones labelled with the following general categories: METADATA, REFERENCES, BODY and OTHER.

Basic structure extraction is implemented by doing character extraction, page segmentation, reading order resolving and zone classification.

The Metadata zone classification is done using SVM where the classification labels are abstract, bib_info, type, title, affiliation, author, keywords, correspondence, dates, and editor. This classification is done using simple rules. The library used is LibSVM for the classification by CERMINE [8].

The CERMINE package for this system is compiled specifically to cater to the problem of extracting information from scholarly literature leaving behind unnecessary classes.

For the system, CERMINE is useful to give us the abstract, author-information and raw text. Apart from extraction of text, it also extracts images from the documents and stores them for later review.



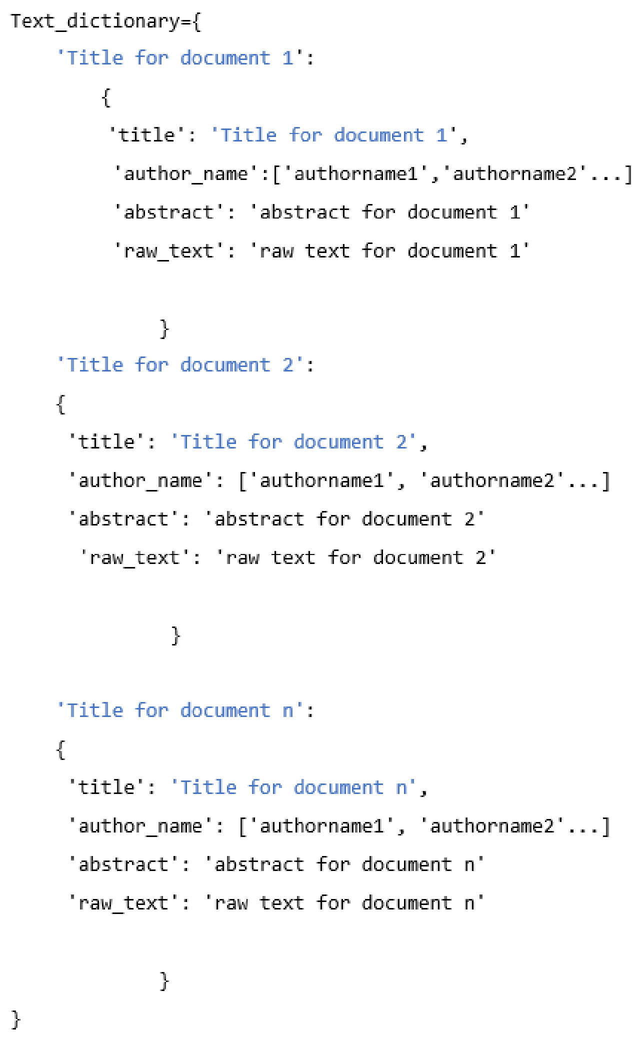
```
object ▶ article ▶
▼ object {1}
    ▼ article {4}
        ▼ front {2}
              journal-meta : value
            ▼ article-meta {3}
                ▼ title-group {1}
                      article-  : A Supervised Approach to Extractive Summarisation
                      title       of Scientific Papers
                ▼ contrib-group {2}
                    ▶ contrib [3]
                    ▶ aff {5}
                ▶ abstract {1}
        ▼ body {2}
            ▶ fig [6]
            ▶ sec [8]
        ▼ back {1}
            ▶ ref-list {1}
          _xmlns:xlink : http://www.w3.org/1999/xlink
```

Fig 5.3 XML output after processing

The above figure represents the tree structure of the XML file generated after extracting text from the PDF file using CERMINE. This XML file has *'object'* as the root node having the *'article'* as the child node. Further *'article'* node is subdivided into *'front'*, *'body'* and *'back'*. The *'front'* element contains the journal information, article meta data including title of the article, name of the authors and their affiliations. The *'body'* node has figures from the document, and textual information stored in the form of sub nodes. The *'back'* node has reference information and is also stored in the form of sub nodes. The images from the document are stored in a local directory with directory name equivalent to the title of the document.

If CERMINE is not able to extract certain information from the document, it would assign a null value to the respective node.

5.3 Processing: After the XML files are generated, the file paths for the same are identified are stored in a list and then passed on to a module name *xml_parser.py* to extract the textual and article information from the document. Iterating over the XML tree using the element tree package in python allows to traverse and extract required information.



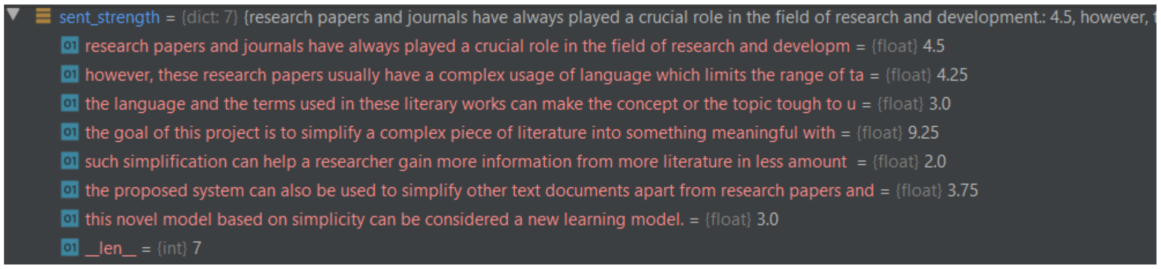
```
Text_dictionary={
    'Title for document 1':
        {
         'title': 'Title for document 1',
         'author_name':['authorname1','authorname2'...]
         'abstract': 'abstract for document 1'
         'raw_text': 'raw text for document 1'

             }
    'Title for document 2':
    {
     'title': 'Title for document 2',
     'author_name': ['authorname1', 'authorname2'...]
     'abstract': 'abstract for document 2'
      'raw_text': 'raw text for document 2'

              }

    'Title for document n':
    {
     'title': 'Title for document n',
     'author_name': ['authorname1', 'authorname2'...]
     'abstract': 'abstract for document n'
     'raw_text': 'raw text for document n'

             }
}
```

Fig 5.4 Document Dictionary after XML processing

A dictionary is defined for every XML file path encountered in the input list. This dictionary then stores information like title, abstract, raw text, author-information. This dictionary is added as value to another dictionary which has the title of the document as the key. This is the main dictionary storing the information from all the input documents and hence serving as the final input for the summarizers. Here few error checks have been implemented for null values stored during Information Extraction phase. If any information like article-information like title and names of the author are not extracted, then the system assigns an 'information could not be extracted' string to that field. This helps to handle runtime exceptions and resume the processing without stopping completely.

5.4 Summarization and Evaluation: The data, stored in the form of document dictionary, is now ready to be passed to the summarizers. Four methods to demonstrate extractive summarization have been implemented. These methods implement specific algorithms or a technique to get an extractive summary. The method implementing spacy package is based on weighted term frequency matrix, Sumy implements Latent Semantic Analysis, GENSIM method involves text rank and NLTK implements TFIDF.

- Weighted term frequency method using spacy: Spacy is an open source natural language processing library by Explosion [9]. It is used for part of speech tagging, named entity recognition and other lexical operations involving text In this implementation , Tokenization of speech is followed by appending nonstop-word tokens having POS (Parts of Speech) as either PROPN, ADJ, NOUN, VERB. This extractive method helps to sustain the named entities thereby increasing the significance of the summary. The concept of weighted frequency is used for extracting sentences. This method is implemented to demonstrate how the frequency of the terms can affect the summarization quality. The input text is tokenized and cleaned before it can be used for summarization. We first create a dictionary for word frequencies by adding terms which are not present in the spacy default stop words list and which are not punctuation marks as well. Once this dictionary is generated from the document, we then use it to calculate weighted term frequency by updating the frequency score of the terms i.e. by dividing the scores with the maximum score. The scores are now scaled between 1 and 0, where 1 represents the term that occurs the most. After calculating the weighted score for each term and updating the values in the dictionary, we use it to replace the words in the sentences with their weighted score which gives an additive score of the weighted term frequency for the sentence. A final dictionary contains sentences as the key and the additive weighted term scores are values. We then select the top sentences with the most occurrences.



```
sent_strength = {dict: 7} {research papers and journals have always played a crucial role in the field of research and development.: 4.5, however, 
    research papers and journals have always played a crucial role in the field of research and developm = {float} 4.5
    however, these research papers usually have a complex usage of language which limits the range of ta = {float} 4.25
    the language and the terms used in these literary works can make the concept or the topic tough to u = {float} 3.0
    the goal of this project is to simplify a complex piece of literature into something meaningful with = {float} 9.25
    such simplification can help a researcher gain more information from more literature in less amount  = {float} 2.0
    the proposed system can also be used to simplify other text documents apart from research papers and = {float} 3.75
    this novel model based on simplicity can be considered a new learning model. = {float} 3.0
    __len__ = {int} 7
```

Fig 5.5 Abstract subjected to weighted frequency-based approach.

The above figure shows the output dictionary having sentences with words less than 30 and their additive scores as the dictionary values.

This method has two input parameters: Document raw text and expected number of sentences in the summary. The expected number of sentences cannot be more than the input length of the raw text.

- Latent Semantic Analysis based summarization using Sumy**:** Sumy is a python package by Miso-Belica [10] for unsupervised extractive text summarization. The package provides four summarizing methods namely Lex Rank, TextRank, LSA and Luhn heuristics.

  Here we are using the LSA summarizer i.e. Latent Semantic Analysis [13]. It is a method for extracting and representing the significant meaning and usage of the words by statistical calculation applied to a large corpus of text. LSA uses a term-document matrix, which has the frequency of the terms in the documents. It is a highly sparse matrix with rows as terms and columns as corresponding documents. We start by declaring a matrix, which has the significance of a term in a sentence, also known as term-sentence matrix. Each column represents the weighted value of the frequency-term vector of the respective sentence for the document under consideration. Suppose there are t terms and s sentences in the document, we can say that we will have a $t \times s$ matrix for a document. This matrix would be sparse in nature. Given this matrix M of $t \times s$, without loss of generality $t \geqq s$, the SVD of M is defined as $M=P \sum Q^T$, where $P =[p_{ij}]$ is $t \times s$ column-orthonormal matrix whose columns are called left singular vectors. $\sum$ is an $s \times s$ diagonal matrix which is diagonal in nature. The diagonal elements are non-negative singular values arranged in reverse order, and Q=[q_{ij}] is an $s \times s$ matrix having orthonormal matrix, the columns are called singular right vectors. The matrix P extracts sentence-to-sentence similarity whereas the matrix Q extracts term-to-term similarity The SVD derives the latent semantic structure from the document represented by matrix M. Sumy implementation for singular vector decomposition is done using *numpy.linalg.singular_vector_decomposition()* which takes a matrix M and gives P, ∑ and Qt as a result. Further rank is calculated for each column value i.e. *s*.

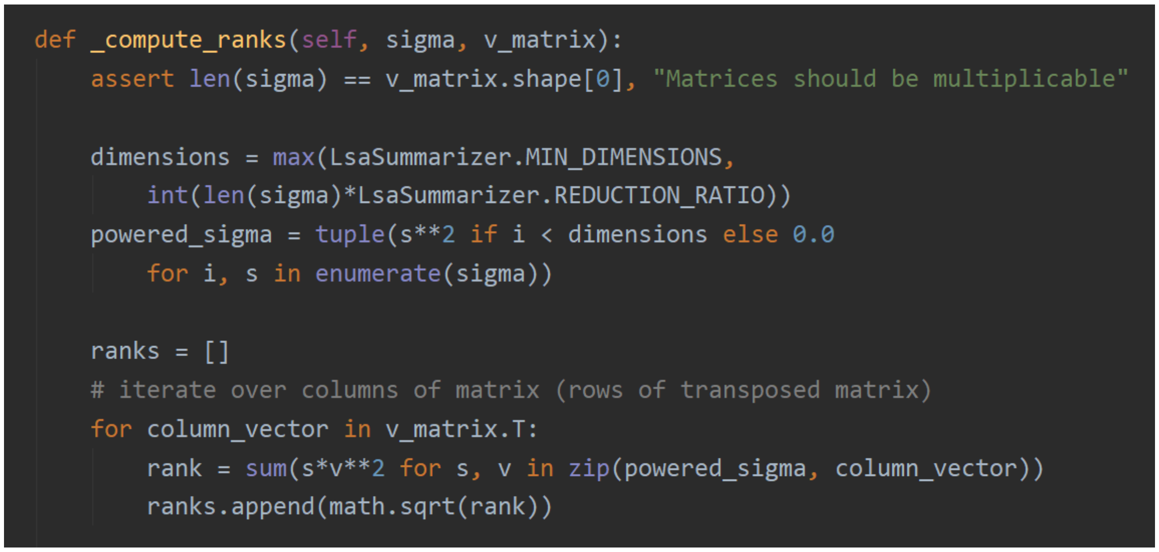

```python
def _compute_ranks(self, sigma, v_matrix):
    assert len(sigma) == v_matrix.shape[0], "Matrices should be multiplicable"

    dimensions = max(LsaSummarizer.MIN_DIMENSIONS,
        int(len(sigma)*LsaSummarizer.REDUCTION_RATIO))
    powered_sigma = tuple(s**2 if i < dimensions else 0.0
        for i, s in enumerate(sigma))

    ranks = []
    # iterate over columns of matrix (rows of transposed matrix)
    for column_vector in v_matrix.T:
        rank = sum(s*v**2 for s, v in zip(powered_sigma, column_vector))
        ranks.append(math.sqrt(rank))
```

Fig 5.6 Rank calculation for LSA from genism module

The number of sentences in the output are arranged based on the calculated ranks.

- Textrank based summarization using Gensim**:** Genism [11] summarizer is based upon a ‘modified’ version of TextRank algorithm [14]. TextRank algorithm is a graph-based algorithm. TextRank is useful for keyword extraction and sentence extraction as well since the algorithm works in a similar way for both.

To apply TextRank, we create a graph related to the text, where the graph vertices are representative for the units to be ranked. For the task of sentence extraction, the goal is to rank entire sentences, and thus a vertex is added to the graph for each sentence within the text. We are defining a 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. Such a relation between two sentences may be seen as a process of "recommendation": a sentence that addresses certain concepts in an exceedingly long text gives the reader a "recommendation" to visit other sentences within the text that address the identical concepts, and thus a link can be established amongst any two such sentences that have common content and context. The similarity measure for the original algorithm is different from the genism implementation [16]. Selection of the sentences to considered for the graph is done keeping in mind the overlapping of the sentences. Normalization factor is given to a group of sentences to avoid selection of long sentences. To calculate normalization factor, we count the length of the overlapping substring and then divide this length with length of that factor. Consider the example of these two sentences:

This is apple - overlapping substring with length as 13

I think ***this is apple,*** *but I am not sure.* 0.325 is the normalization factor.

I think ***this is apple*** *but cannot be sure about it*. 0.26 is the normalization factor.

Here the sentence with high normalization score would be considered. After the selection on the sentences the similarity score is calculated using BM25 [17].

$$BM25(R,S) = \sum_{i=1}^{n} IDF(s_i) \cdot \frac{f(s_i,R)\cdot(k_1+1)}{f(s_i,R)+k_1\cdot(1-b+b\cdot\frac{|R|}{avgDL})}$$

R and *S* are the sentences for which the similarity is calculated, s_i is the *i* th term in the sentence R, $f\ (s_i\ ,R)$ is the frequency of the term s_i in the sentence R. The parameters k_1 and b are parameters having the value 1.5 and 0.75, respectively. The average length of the sentences in the text is denoted by *avgDL*. IDF(s_i) represents inverse document frequency for term s_i.

The definition for the above functions shows that if a term is in more than 50% of the document, it will start taking up negative values. To avoid this, we correct the IDF

$$IDF(s_i) = \begin{cases} log(N-n(s_i)+0.5) - log(n(s_i)+0.5) & \text{if } n(s_i) > N/2 \\ \varepsilon \cdot avgIDF & \text{if } n(s_i) \le N/2 \end{cases}$$

Here N is the number of words in the document. And ε is a constant which takes the value 0.25 for the implementation in genism. The similarities calculated are now used to create a fully interconnected graph. This weighted graph is now subjected to a ranking algorithm like that of PageRank and finally the output is sorted to maintain the order of the original input.

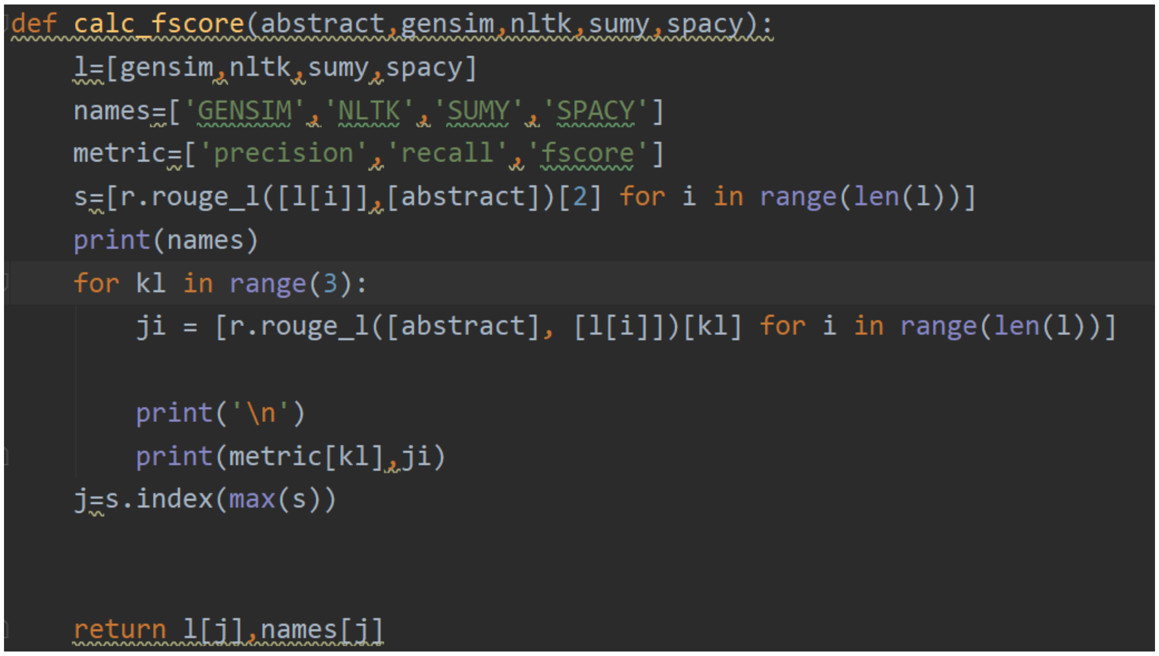
TextRank is ideal for projects involving entire sentences, since it allows for a ranking over text units that is recursively computed based on information drawn from the entire text.

- TF-IDF based summarization using NLTK: Natural Language Toolkit [14] based summarizer uses TF-IDF [15] for extracting sentences. TF-IDF or Term Frequency-Inverse Document Frequency, is an algorithm which is divided into two parts – calculating the term frequency and then multiply it by the inverse document frequency. Term frequency is finding out how common a word is in a document by taking the ratio of the number of times a word appears in a document and the total number of words in a document. Inverse document frequency is the exact opposite of term frequency. It finds out how rare a word is within a document. A term-frequency dictionary is created in which the terms are the ones which do not belong in stop words, which is followed by tokenization. Once the term-frequency dictionary is generated, it is stored as a dictionary with the sentences as key and dictionary of words and their occurrences. For tokenizing the sentences, sent tokenize() is used and once that is done we then create a frequency matrix like we did for generating weighted term frequency matrix. The difference here would be that each sentence is a key and the corresponding value would be a dictionary of term frequency term frequency. The TF(t) is the number of times a term t appears in the text divided by total number of terms in a text. Next thing would be to calculate a matrix which represents number of sentences that contain a specific term 't'. This is done to in order to calculate IDF(t) which is logarithmic function of total number of sentences in the text divided by the number of sentences containing term 't' in it. The product of both the matrix gives the sentence scores and these sentences which have score higher than the threshold will the be selected for summary. For this implementation, average value of the sentence score has been considered as the limiting function.

The above four systems are just to demonstrate different approaches and their summary results. The output for the above systems may vary for different texts because of their nature of operation. The reason for showcasing four different approaches is to demonstrate that the extraction process varies when we use different approaches. These intermediary outputs will now be fed to an evaluation model where we will compare the summaries generated with the abstract to finalize the best one.

Evaluation**:** Once the summarizations are generated with the above four methods, the selection of one good summary is to be done. Evaluating the summarization methods is important as the quality of the extract depends upon the correlation with the subject. The most common summarization metrics used for extractive summarization are content based evaluations as described in *"Evaluation Measures* *of Text Summarization"* [18]. The system implements the n-gram matching evaluation technique. This includes the ROUGE family of measures, which are based on the similarity of n-grams [19], first introduced in 2003. The evaluation metric used here is ROUGE metrics. ROUGE is an abbreviation for Recall-Oriented Understudy for Gisting Evaluation. It consists of metrics to implicitly decide the quality of a summary by comparing it to other (ideal) summaries generated by humans. Since the summarization process is unsupervised and does not have a reference summary generated by human, we consider abstract as a true summary of the whole document. This idea is loosely based upon AbstractRouge [5], a metric discussed by Ed Collins, Isabelle Augenstein and Sebastian Riedel in their research paper. The idea here is to select a summary which has a high ROUGE-L score with the abstract. Once the ROUGE-L score is calculated, the summary with maximum F-score is selected, which makes it a relevant extract from the document.

ROUGE-L score relies on matching longest common subsequence. The output with the highest F-score for the rouge score is then used for further processing. The reference summary is usually a summary generated by humans but since our system is unsupervised, we make use of the abstract.



```
def calc_fscore(abstract,gensim,nltk,sumy,spacy):
    l=[gensim,nltk,sumy,spacy]
    names=['GENSIM','NLTK','SUMY','SPACY']
    metric=['precision','recall','fscore']
    s=[r.rouge_l([l[i]],[abstract])[2] for i in range(len(l))]
    print(names)
    for kl in range(3):
        ji = [r.rouge_l([abstract], [l[i]])[kl] for i in range(len(l))]

        print('\n')
        print(metric[kl],ji)
    j=s.index(max(s))


    return l[j],names[j]
```

Fig 5.7 Code snippet to calculate the ROUGE-L metrics

ROUGE-L determines the longest matched sequence of words using the longest common subsequence (LCS). The good part about using LCS is that it does not require consecutive matches, but in-sequence matches that reflect sentence level word order. Since it automatically includes longest in-sequence common n-grams, you do not need a predefined n-gram length. F-score considers recall and precision both. F-score is the harmonic mean of precision and recall values. F-score ranges from 0 to 1, with 1 being the best F-score thereby having best recall and precision values. Hence, F-score for ROUGE-L metrics is best for comparing the extract with the abstract of the document.

The figure below illustrates how the output from our information extraction block, which is in the form of raw text, is summarized and passed on for evaluation. This process is done for each document

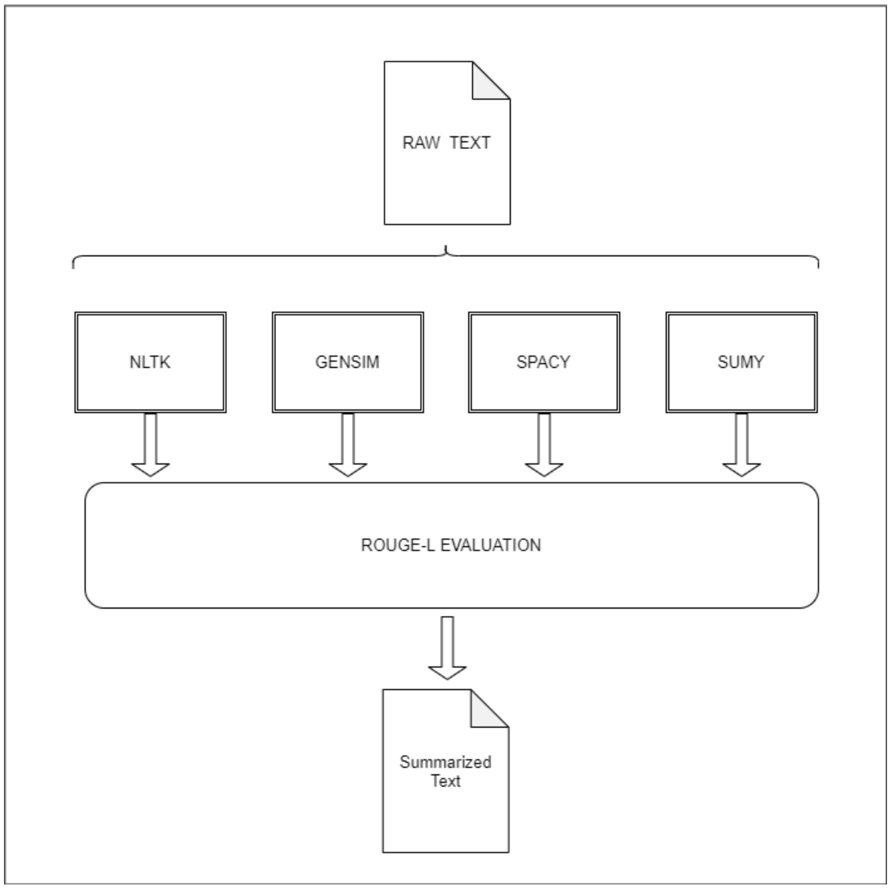


Fig 5.8 Summarization and Evaluation

5.5 Simplification: Once the extract is finalized after the evaluation, we further subject it to sentence simplification to make it easier to read. The summary after simplification should be simple and should not have complex grammatical sentences. The simplification part is for better readability and therefore we have two options for the process.

- Context breakdown: Context analysis for text is important as it helps to break down a sentence into a simpler group of sentences. In natural language processing, parts of speech help to understand the context from a structured text.

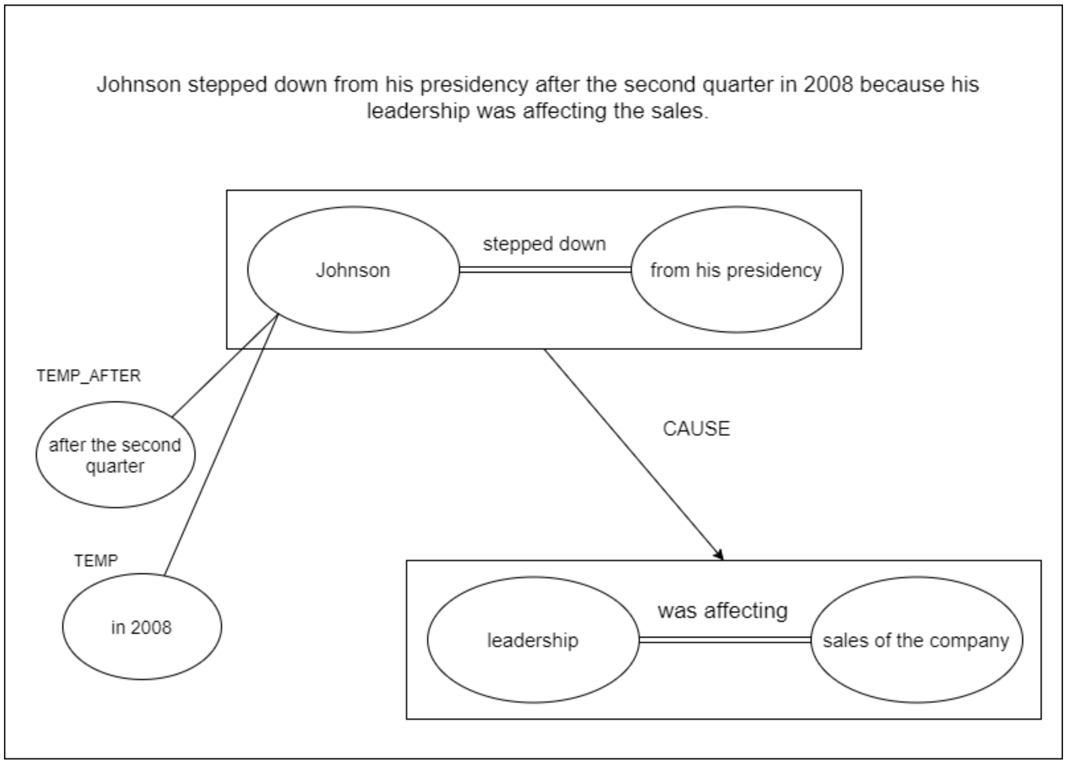
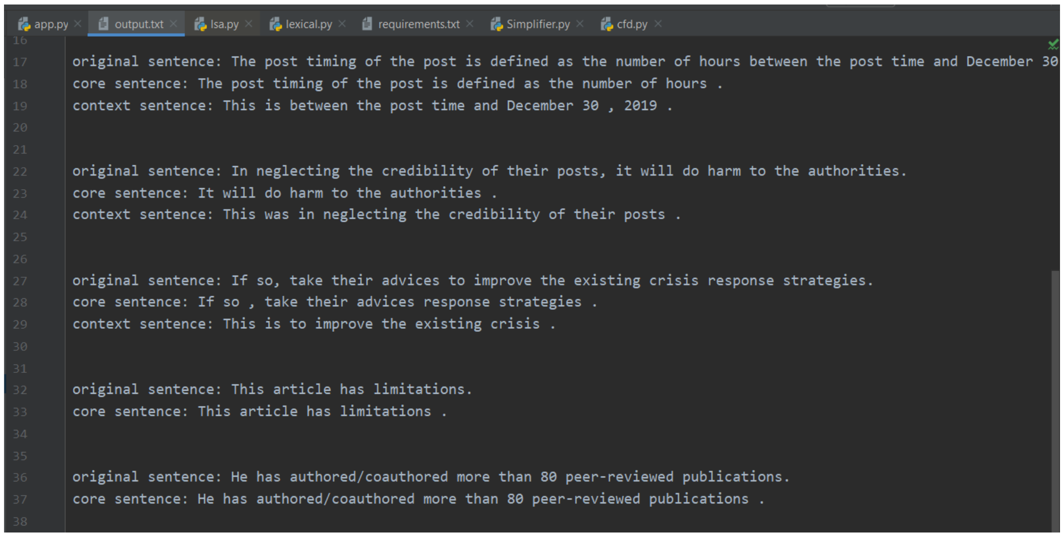


Fig 5.9 Context Analysis

Context analysis consists of resolving co-references from a sentence. Understanding sentence structure and their components like appositive phrases, containing subordinations, coordination is useful for transforming into a simpler structure. Identification of rhetorical relationships between the sentences and extracting binary relationships like subject and predicate from each sentence helps understand the granularity of a sentence. The above attributes of a sentence help to establish a context relationship and thereby helps in generating simple sentences.

The above figure illustrates how a sentence can be broken down to find the context. To breakdown the evaluated summary we make use of an open source sentence simplification java program which was originally implemented in the graphene pipeline by *Lambda*^{3} [20]. Once the context breakdown is done for each sentence, we print the core sentence followed by the context of the sentence. Similarly, the whole text gets broken down into much simpler sentences. The number of sentences in the output are increased due to breakdown of the original sentences.

StanfordCoreNLP [21] is used by the sentence simplification library for context breakdown of the input sentences. The library is used within the compiled JAR file. At first, tokenization is done using in-built PTB Tokenizer – default tokenizer for the library. After tokenization, we use the 'ssplit()' function to get the sentences after splitting. These sentences are then fed to the LexParser class, which uses the pre-trained englishPCFG.ser.gz model to derive the structure of the sentence, following which, POS tagger from class Maxent is used to get the parts of speech tags for each word in the sentences. Finally, AbstractSequenceClassifier loads english.all.3class.distsim.crf.ser.gz model to classify core sentences.



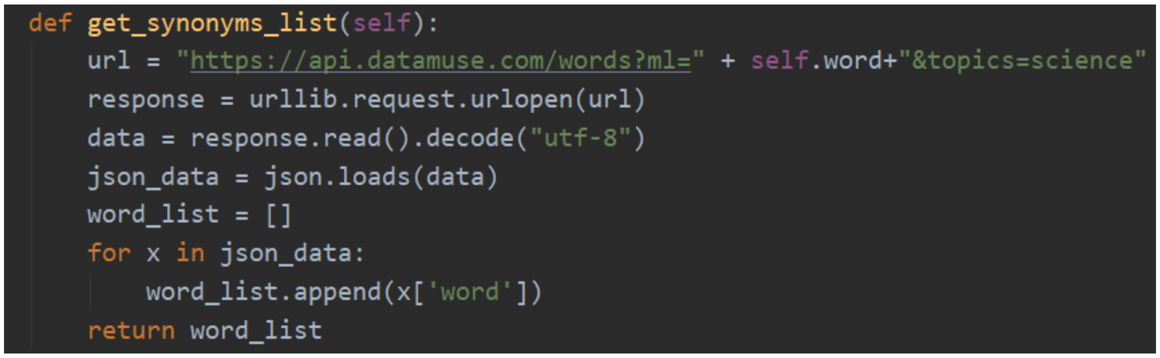
```
original sentence: The post timing of the post is defined as the number of hours between the post time and December 30
core sentence: The post timing of the post is defined as the number of hours .
context sentence: This is between the post time and December 30 , 2019 .


original sentence: In neglecting the credibility of their posts, it will do harm to the authorities.
core sentence: It will do harm to the authorities .
context sentence: This was in neglecting the credibility of their posts .


original sentence: If so, take their advices to improve the existing crisis response strategies.
core sentence: If so , take their advices response strategies .
context sentence: This is to improve the existing crisis .


original sentence: This article has limitations.
core sentence: This article has limitations .


original sentence: He has authored/coauthored more than 80 peer-reviewed publications.
core sentence: He has authored/coauthored more than 80 peer-reviewed publications .
```

Fig 5.10 output.txt for each document for context breakdown

The output comprises of three subparts – original sentence, core sentence and context. If the sentence is not able to identify any core or context, it would just return the original sentence. Ideally the system displays the core sentence along with the context sentence, however we can customize the output as needed.

- Article-Rewriting: Another approach to simplify a text would be to simplify the complexity of the text by replacing the complex words with simpler words. This technique is also known as lexical simplification. For this approach, complex words are replaced with synonyms from an online dictionary. We make use of the Datamuse API [22] for finding synonyms for several parts of speech from a sentence. Datamuse API is a word query engine for developers. Its main usage is to find words that match a given set of rules and criteria.

There are many criteria options available for the API usage, but we are concerned with 'ml=' and 'topic=' endpoints of this API. We use this API to fetch synonyms from the dictionary available online from the Datamuse database, which is based on Google Books Ngrams. The ‘topic=’ parameter acts just as a hint in the HTTP request call. The results of this method can be vague as the lexical simplification done over a dictionary which is not domain specific can change the overall meaning and loss in the information. Also, this approach requires high bandwidth and is not scalable as it has high latency for response time.



```
def get_synonyms_list(self):
    url = "https://api.datamuse.com/words?ml=" + self.word+"&topics=science"
    response = urllib.request.urlopen(url)
    data = response.read().decode("utf-8")
    json_data = json.loads(data)
    word_list = []
    for x in json_data:
        word_list.append(x['word'])
    return word_list
```

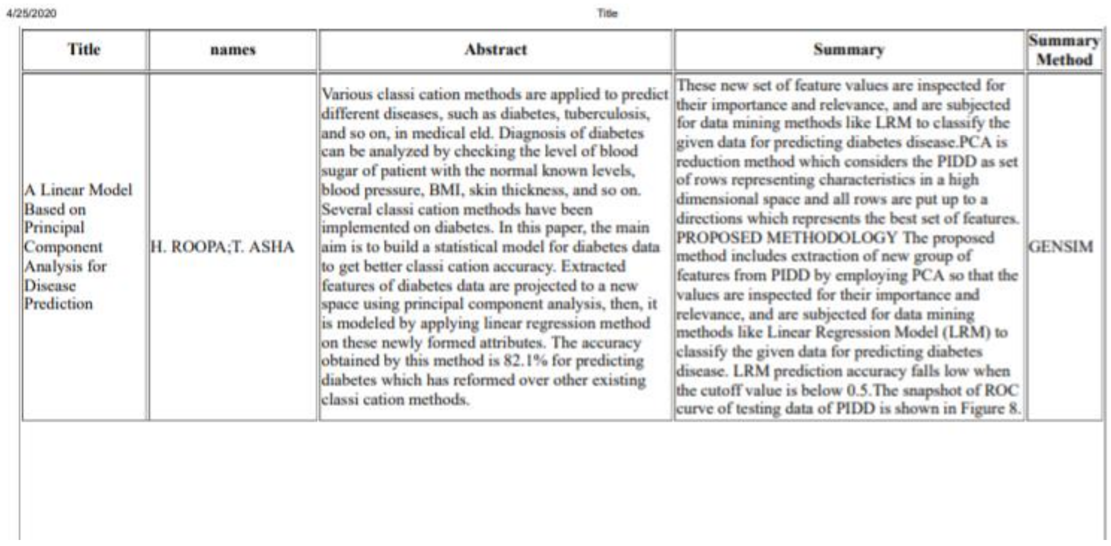
Fig 5.11 Datamuse API call

This method can be improved by implementing a topic-based dictionary for the document. One way to deal with this situation is by implementing article classification and creating a dictionary for possible topics. To showcase this lexical-simplification, medical science subject was selected for implementation and a publicly available dictionary from a repository known as SimpleScience [23] was used as dictionary to substitute complex words identified as a part of subject jargon. The lexical simplification implemented using this dictionary is very specific to ‘medical sciences’ subject.

Chapter 6

Observations and Analysis

The final output of the system is a consolidated document having summaries presented in a tabular format. The output is an HTML table with column names as title of the document, names of the authors, abstract of the document, final output summary and the type of summarization method selected after evaluation.



4/25/2020 Title

Title	names	Abstract	Summary	Summary Method
A Linear Model Based on Principal Component Analysis for Disease Prediction	H. ROOPA;T. ASHA	Various classi cation methods are applied to predict different diseases, such as diabetes, tuberculosis, and so on, in medical eld. Diagnosis of diabetes can be analyzed by checking the level of blood sugar of patient with the normal known levels, blood pressure, BMI, skin thickness, and so on. Several classi cation methods have been implemented on diabetes. In this paper, the main aim is to build a statistical model for diabetes data to get better classi cation accuracy. Extracted features of diabetes data are projected to a new space using principal component analysis, then, it is modeled by applying linear regression method on these newly formed attributes. The accuracy obtained by this method is 82.1% for predicting diabetes which has reformed over other existing classi cation methods.	These new set of feature values are inspected for their importance and relevance, and are subjected for data mining methods like LRM to classify the given data for predicting diabetes disease.PCA is reduction method which considers the PIDD as set of rows representing characteristics in a high dimensional space and all rows are put up to a directions which represents the best set of features. PROPOSED METHODOLOGY The proposed method includes extraction of new group of features from PIDD by employing PCA so that the values are inspected for their importance and relevance, and are subjected for data mining methods like Linear Regression Model (LRM) to classify the given data for predicting diabetes disease. LRM prediction accuracy falls low when the cutoff value is below 0.5.The snapshot of ROC curve of testing data of PIDD is shown in Figure 8.	GENSIM

Fig 6.1 Output for one pdf

During the evaluation phase, we go through a few metrics for the generated summary. We have considered 20 articles for generating summaries. For every research article, we calculate the ROUGE-L score, precision, recall and F-score. This is done for each type of summary generated using different methods i.e., GENSIM, NLTK, SUMY and SPACY. The best summary is selected by using the maximum F-score for all the four methods.

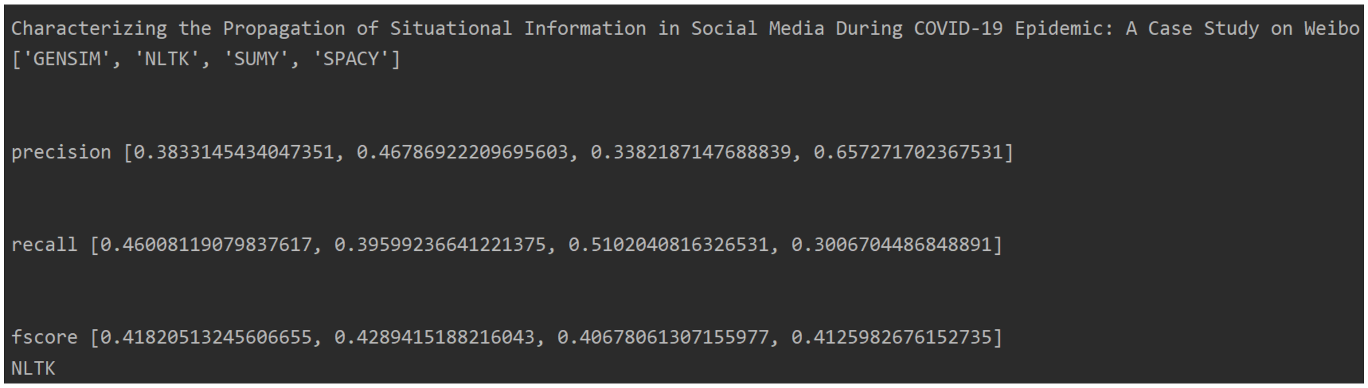
6.1. F-score analysis:

In Statistics, for binary classification, F-score is a metric used to test a test's accuracy. It includes – precision and recall of the test to calculate the score where precision is the number of accurate positive results divided by the number of all known negative results returned by the model while recall is the number of accurate positive results divided by the number of samples having relevance (samples that have been identified as positive). The F-score is the sub-contrary mean of precision and recall.

We use a module – PyRouge [24], a Python wrapper for the ROUGE summarization evaluation package. The F-score is calculated using the following formula:

$$F_\beta = (1 + \beta^2) \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{(\beta^2 \cdot \mathrm{precision}) + \mathrm{recall}}.$$

The F(beta) score "measures the effectiveness of retrieval with respect to a user who attaches beta times as much importance to recall as precision" [25].



```
Characterizing the Propagation of Situational Information in Social Media During COVID-19 Epidemic: A Case Study on Weibo
['GENSIM', 'NLTK', 'SUMY', 'SPACY']


precision [0.3833145434047351, 0.46786922209695603, 0.3382187147688839, 0.657271702367531]


recall [0.46008119079837617, 0.39599236641221375, 0.5102040816326531, 0.3006704486848891]


fscore [0.41820513245606655, 0.4289415188216043, 0.40678061307155977, 0.4125982676152735]
NLTK
```

Fig 6.2 ROUGE-L metrics

In the above figure, we observe that 0.4289 is the maximum value from the array of F score for one of the papers and thereby we select the corresponding summary.

6.2 Heatmap Analysis:

Every document gives a three by four matrix when subjected to ROUGE_L metrics. The three rows in the matrix represent precision, recall and f-score respectively while the four columns represent the summarization methods, which are - Text Rank using GENSIM, TFIDF using NLTK, LSA using SUMY and Weighted frequency-based ranking using SPACY.

To observe the metric values for the documents, we select four matrices with different types of summarization method selected based on the maximum f-score. The observations made from these heat maps helps in analyzing the summarization methods in more detail.

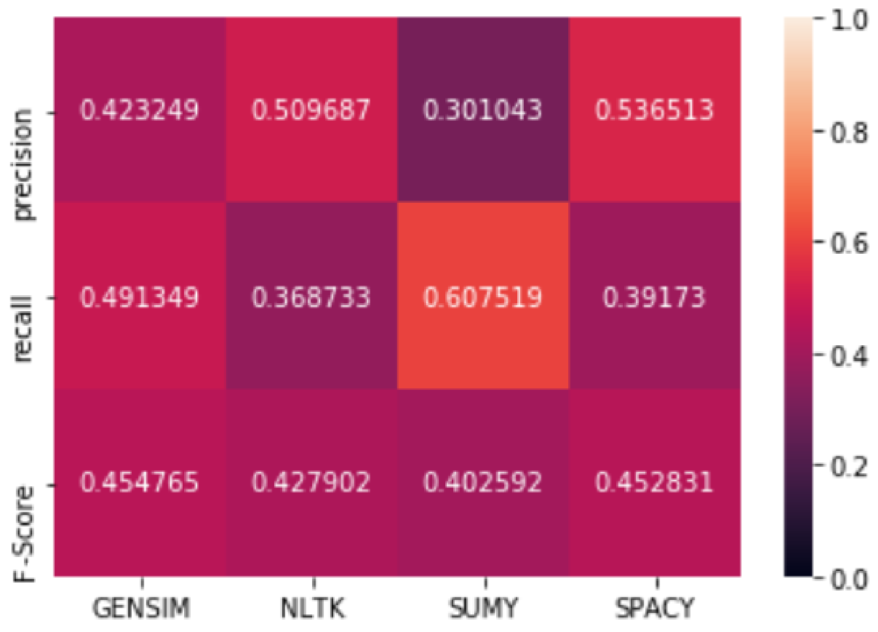


Fig 6.3 GENSIM summary with high F-score

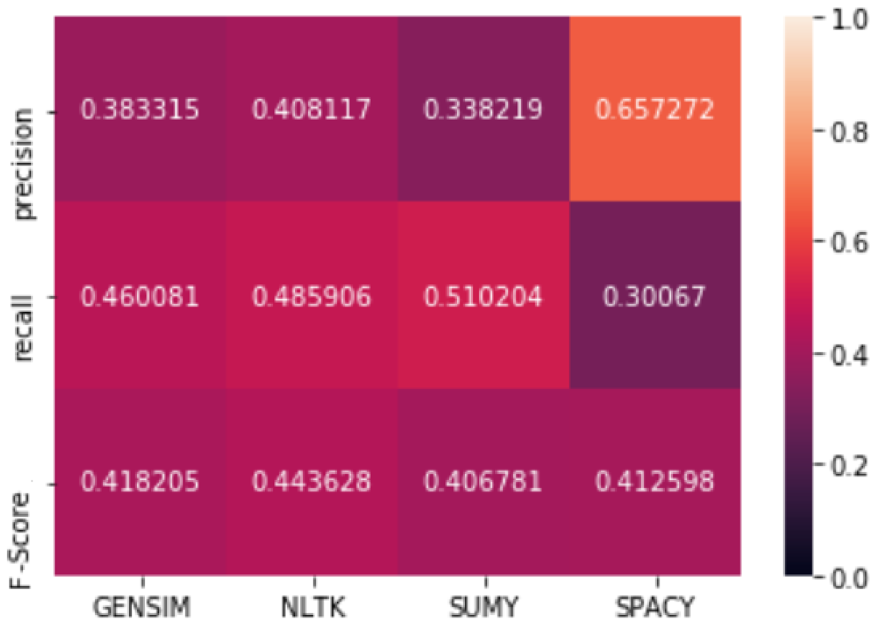


Fig 6.4 NLTK summary with high F-score

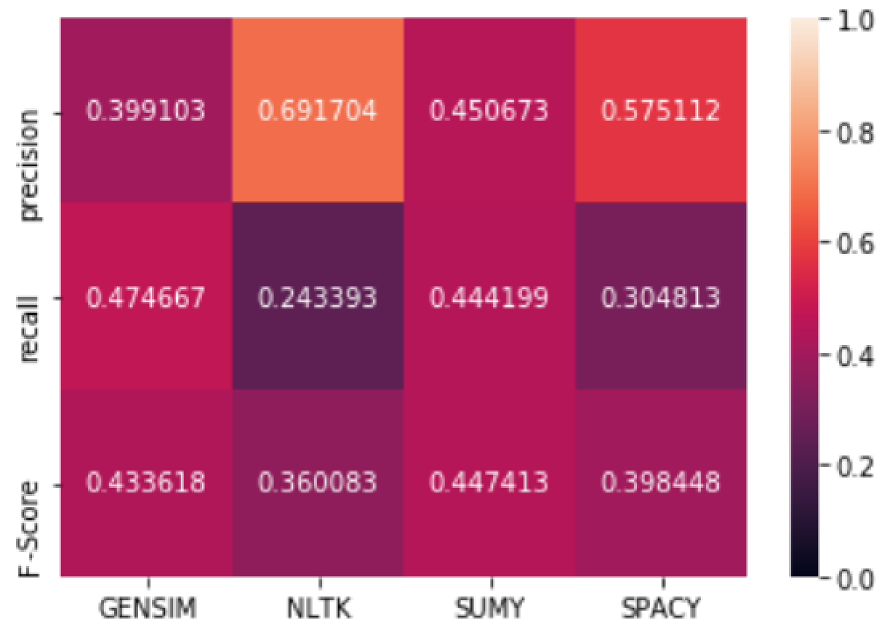


Fig 6.5 SUMY summary with high F-score

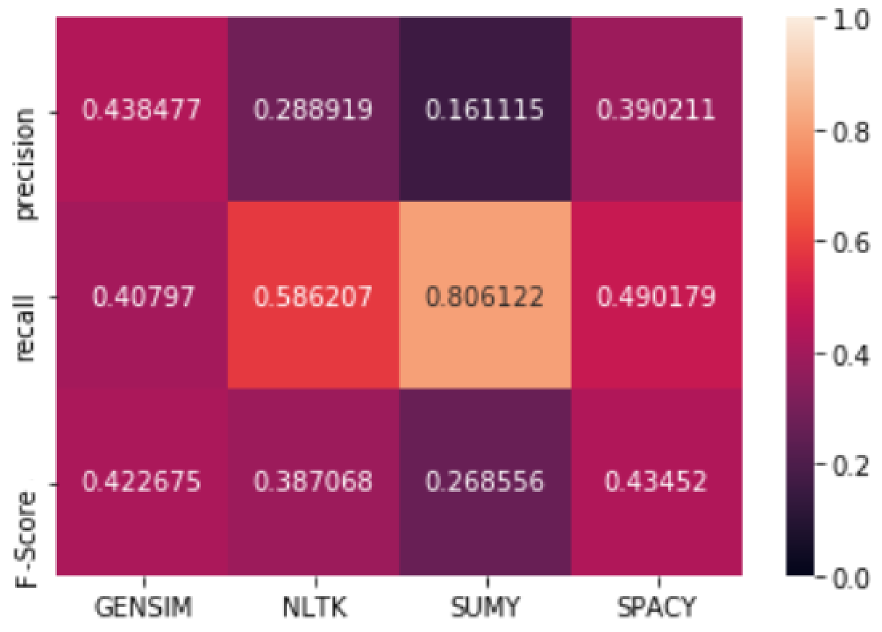


Fig 6.6 SPACY summary with high F score

The documents chosen for each kind of summarization methods were random. This was done to observe any kind of anomaly in the metric measurement. The heatmaps were generated using the 'Seaborn' package in a Jupyter Notebook. As we can observe in the above figures, recall value for summaries generated by Latent Semantic Analysis using Sumy are higher in most cases. The summaries generated by Text Rank using GENSIM tend to have high F-scores in most cases. This was observed for most of the documents in the dataset. Summaries generated by the weighted frequency method using Spacy have a high precision value and that is mainly because of the significance of weight matrix for the document.

6.3 Runtime Analysis:

The evaluation is done based on the assumption that abstract is an ideal summary of the article. Analysis is done on a sample of 20 research papers. The selection of these documents is random and are from different publications like IEEE, aRXiv, Jetir and others. The total size of this dataset of documents is 39.5 Megabytes. The smallest size being 93 Kilobytes and largest one being 17.8 Megabytes.

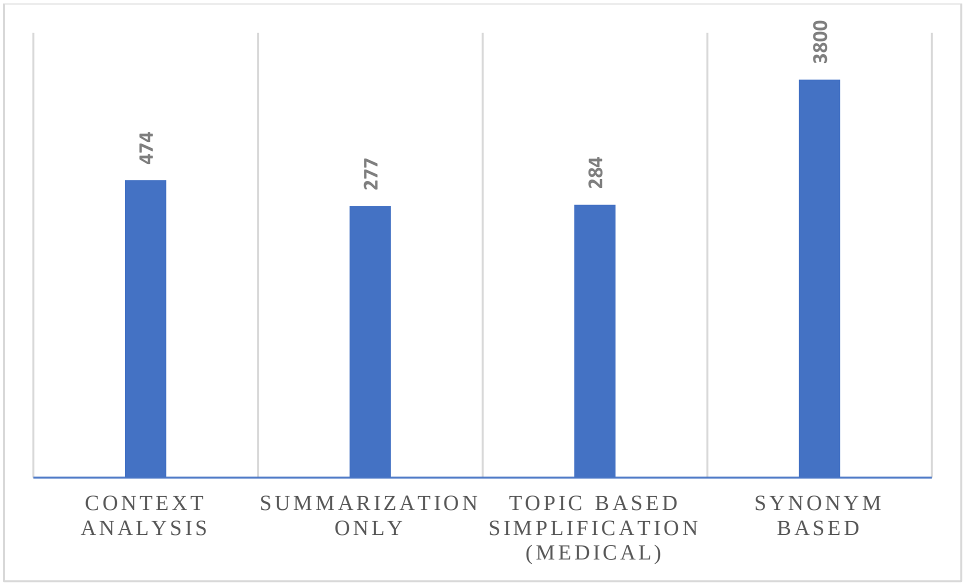


Fig 6.7 Runtime for 20 papers (in seconds)

The above figure shows the runtime to generate final output in the form of an HTML table. The execution was done on a system running Windows 10 operating system with 16 Gigabytes of physical memory and i5-8300H clocked at 2.30 Gigahertz. It is observed that synonym based lexical simplification takes a huge amount of time since it fetches synonyms for each parts of speech tagged words. Also, the technique is not efficient as it requires an HTTP API call for every word and hence increases the runtime significantly. Therefore, the technique is good as just a proof of concept.

6.4. Survey Analysis:

Apart from this, an anonymous public survey was conducted with the help of Google forms. A link was shared on social media platforms to get insights regarding the summarization of the article. A web document with extractive summaries for the twenty documents were distributed amongst family members, friends, and peers in acquaintance along with a Google form link. These responses were recorded in an excel sheet and were reviewed for evaluation. The form collected three attributes - readability, relevance of the summary and understanding of the subject on the scale of 1 to 10, with 10 being the highest. A total of 76 responses were collected using this medium and following are the results:

Readability (%)	Relevance (%)	Understanding (%)
72.10	68.86	68.84

Table 6.1 Average value of 76 responses

The above results show that readability factor for the document is high but relevance and understanding attribute is comparatively lower. The relevance and understanding attributes have almost same value as the participant might have misunderstood the attributes.

Chapter 7

Future Work

The system meets the requirement of a user-friendly tool to span scholarly literature in an easy way. The convenience of summarizing and simplifying scholarly literature makes it a useful tool for researchers and students. It has a very high potential to achieve much more functionalities and can be proven more useful. Following are some future goals for the system:

- Recommending literature based on sematic relationship: The system can be further improved by having suggestions and recommendations for similar literature for better understanding of the topic.
- Usage of BERT [26], a pre-trained transformer model for natural language processing, can be used to improve the performance of the extractive summarization. This approach can help increase the ROUGE score for the extracted summary.
- Adding Cross Entropy Summarization [27] technique for better yield of information from the documents. This approach has proven to be a good implementation for unsupervised, query focused multi-document summarization method. It does not require any kind of domain knowledge.
- Implementing the LEXenstein [28] library for text simplification and better context breakdown of sentences. This will help over come the issue of lexical simplification without the need of corpus. LEXenstein pipeline involves identification of complex

words, substitution generation, ranking of the substitution and then simplifying the text.

- Using seq2seq framework [29] to attain a better production level result. The implementation of seq2seq framework for abstractive summarization along with extractive summarization can be used to fine tune the performance of the situation.
- Implement containerization to help the system scale. This will help in scaling and distributing the application once it is deployed on a production cover.
- Adding session management to allow parallel access to the application. The implementation is currently in development phase; therefore, it implements serialization.
- Improving the capability to read all kinds of literature, irrespective of the structure of the document.
- Making a REST application programming interface (API). This will improve and broaden the use cases for the system.
- Mobile application development, using the powerful capabilities of the current generation hardware, will improve the reach of the system along with increasing convenience for the users.

Chapter 8

Conclusion

After performing several experiments and analysis on the output generated by the system, we can conclude that the proposed system for extractive summarization and simplification of scholarly literature is a good way to obtain extractive summarization. While performing analysis on a dataset of twenty papers, it is observed that eleven out of twenty papers were summarized using the text rank algorithm implemented by gensim. This supports the fact that gensim is the most evident method to generate summary as per the observations.

The system performs great when it comes to simplifying the summary using context breakdown, but it needs a better approach for lexical simplification. The extractive summarization and simplification using context break is the ideal way to use this system. The consolidated table generated for the documents gives a fair insight to a user with an average rating of 7.2, 6.8 and 6.8 for readability, relevance and understanding, respectively. To use this system in the best way possible, one should input similar documents having a specific structure. This way one can read similar documents in an easier way.

References

[1] Richard Feynman, Wikipedia Article; https://en.wikipedia.org/wiki/Richard_Feynman

[2] H. P. Luhn, "The Automatic Creation of Literature Abstracts," in *IBM Journal of Research and Development, vol. 2, no. 2*, pp. 159-165, Apr. 1958.

[3] Barrios, F., López, F., Argerich, L., Wachenchauzer, R., "Variations of the Similarity Function of TextRank for Automated Summarization" in *Anales de las 44JAIIO. Jornadas Argentinas de Informática, Argentine Symposium on Artificial Intelligence, 2015*.

[4] Guy Feigenblat, Odellia Boni, Haggai Roitman, and David Konopnicki. 2017. “SummIt: A Tool for Extractive Summarization, Discovery and Analysis.”. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM ’17)*. Association for Computing Machinery, New York, NY, USA, 2459–2462. DOI: https://doi.org/10.1145/3132847.3133183

[5] Ed Collins, Isabelle Augenstein, Sebastian Riede., l A Supervised Approach to Extractive Summarization of Scientific Papers, 2017, arXIv. DOI: https://arxiv.org/abs/1706.03946v

[6] Michihiro Yasunaga ,Jungo Kasai, Rui Zhang ,Alexander R. Fabbri ,Irene Li and Dan Friedman, Dragomir R. Radev. “ScisummNet: A Large Annotated Corpus and Content Impact Models for Scientific Paper Summarization with Citation Networks”, 2019, arXIv. DOI: https://arxiv.org/abs/1909.01716

[7] Wen Xiao ,Giuseppe Carenini “Extractive Summarization of Long Documents by Combining Global and Local Context” ,2019, DOI: https://arxiv.org/abs/1909.08089

[8] Dominika Tkaczyk, Pawel Szostek, Mateusz Fedoryszak, Piotr Jan Dendek and Lukasz Bolikowski. “CERMINE: automatic extraction of structured metadata from scientific literature”. In *International Journal on Document Analysis and Recognition, 2015, vol. 18, no. 4*, pp. 317-335, doi: 10.1007/s10032-015-0249-8.

[9] Honnibal, Matthew and Montani, Ines. “spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing”; https://github.com/explosion/spaCy

[10] BELICA, Michal., “SUMY”; http://hdl.handle.net/11012/53529; https://miso-belica.github.io/sumy/

[11] Radim \v Reh\r u\v rek, and Petr Sojka 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (pp. 45–50). ELRA.; https://pypi.org/project/gensim/

[12] Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc; https://www.nltk.org/

[13] Josef Steinberger, Karel Ježek “Using Latent Semantic Analysis in Text Summarization and Summary Evaluation”; http://www.kiv.zcu.cz/~jstein/publikace/isim2004.pdf

[14] Mihalcea, P. 2004. “TextRank: Bringing Order into Text.” In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (pp. 404–411). Association for Computational Linguistics. https://www.aclweb.org/anthology/W04-3252

[15] Bruno Stecanella “What is TF-IDF?”, 2019; https://monkeylearn.com/blog/what-is-tf-idf/

[16] Federico Barrios and Federico L\'opez and Luis Argerich and Rosa Wachenchauzer 2016. Variations of the Similarity Function of TextRank for Automated Summarization. CoRR, abs/1602.03606; http://arxiv.org/abs/1602.03606

[17] Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M., Gatford, M.: Okapi at TREC-3. In: Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994. pp. 109–126 (1994); http://trec.nist.gov/pubs/trec3/papers/city.ps.gz

[18] Steinberger, Josef, and Karel Ježek. "Evaluation measures for text summarization." *Computing and Informatics* 28.2 (2012): 251-275.

[19] Lin, C.Y. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out* (pp. 74–81). Association for Computational Linguistics.; https://www.aclweb.org/anthology/W04-1013

[20] Andre Freitas, Bernhard Bermeitinger, Christina Niklaus, Matthias Cetto, Siegfried Handschuh. “SentenceSimplifcation by Lambda3”; https://github.com/Lambda-3/SentenceSimplification

[21] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. “The Stanford CoreNLP Natural Language Processing Toolkit”, In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60. [pdf] [bib]

[22] Datamuse.com. 2020. *Datamuse API*. [online] Available at: https://www.datamuse.com/api/

[23] Kim, Yea-Seul, et al. "Simplescience: Lexical simplification of scientific terminology." *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016; https://github.com/yeaseulkim/SimpleScience

[24] Benjamin Heinzerling, Anders Johannsen ‘PyRouge’; https://pypi.org/project/pyrouge/

[25] [Chinchor, 1992] Nancy Chinchor, “MUC-4 Evaluation Metrics, in Proc. of the Fourth Message Understanding Conference”, pp. 22–29, 1992. https://dl.acm.org/citation.cfm?id=1072067

[26] Yang Liu. (2019). “Fine-tune BERT for Extractive Summarization”; https://arxiv.org/pdf/1903.10318

[27] Guy Feigenblat, Haggai Roitman, Odellia Boni, and David Konopnicki. "Unsupervised Query-Focused Multi-Document Summarization using the Cross-Entropy Method". In Proceedings of the *40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 961–964. DOI: https://doi.org/10.1145/3077136.3080690

[28] Paetzold, Gustavo & Specia, Lucia. (2015). "LEXenstein: A Framework for Lexical Simplification." 85-90. 10.3115/v1/P15-4015.

[29] Britz, D., Goldie, A., Luong, T., and Le, Q. 2017. "Massive Exploration of Neural Machine Translation Architectures." *ArXiv e-prints*