CHAPTER - 1

INTRODUCTION

1.1 GENERAL INTRODUCTION

These days, a lot of research papers are published, and the number of materials available also grows. Searching for materials in our field of interest is a difficult undertaking. Summarization effectively communicates a research paper's information without diluting its originality, which facilitates the discovery of papers pertinent to our field of interest. The text's summary is provided by the keywords and keyphrases. A keyword identifies a single word (e.g., computer, keyboard), whereas a keyphrase indicates a multi-word lexeme (e.g., computer science, engineering, machine learning) . An application of document classification, summarizing, etc. is keyword extraction. The selection of the document to read assists with the extraction of pertinent keywords . It is time-consuming to manually extract keywords and keyphrases from documents.

The human extraction of keywords requires a lot of work and takes a long time; for scientific papers published in a single day, it is nearly impossible to manually extract keywords due to their bulk. We suggested a technique that extracts keywords from academic writing in order to swiftly examine the impacts of context word embedding. While it is simple to extract keywords and key phrases from a corpus of big sentences, it can be more challenging to do so from a corpus of shorter sentences. In our suggested method, the contextual word embedding model will successfully extract keywords and important phrases contextually and semantically. The suggested method outperforms some conventional techniques by extracting keywords and concentrating on contextual features.

Finding the words and phrases that best describe a document's primary themes is known as keywords extraction. The usage of relevant keywords in many NLP applications, such as document indexing, text classification, and summarization, has been demonstrated. Keywords extraction has drawn more attention in recent years, and multiple benchmark datasets are now accessible in a variety of fields and languages. However, there aren't many tools available for

automatic keyphrase extraction that can incorporate modern, cutting-edge techniques or be used for quick prototyping, like the Python-based Natural Language Toolkit (nltk) does.

We looked at several current keyword extraction techniques. Numerous techniques and strategies have been developed for keyword extraction from documents. Some of them are Term frequency,support vector machine and other methods have all shown to produce better outcomes when extracting keywords.. All showed improved outcomes when extracting keywords, but they must be correctly recorded to strengthen the connection between words and phrases. We think that by using our approach, we can extract keywords from texts more effectively.

1.2 PROBLEM STATEMENT:

The volume of the Web is increasing exponentially day by day. Such a search query of a particular topic results in various articles, research papers and documents. Out of which, most of the documents are irrelevant with respect to the query. It has been observed that the title or the headline of that document is generally too noticeable. However, when the document is read in detail, its context doesn't relate with the title. Such documents either contain insufficient or irrelevant information with respect to the aimed query. This observation derived the problem statement of the project i.e., to identify the core keywords of the documents which describes the document in the best possible way. Further, the identified keywords will be compared to the mentioned keywords of the documents to know the relevance of the document with their mentioned keywords.

1.3 SIGNIFICANCE OF THE PROBLEM:

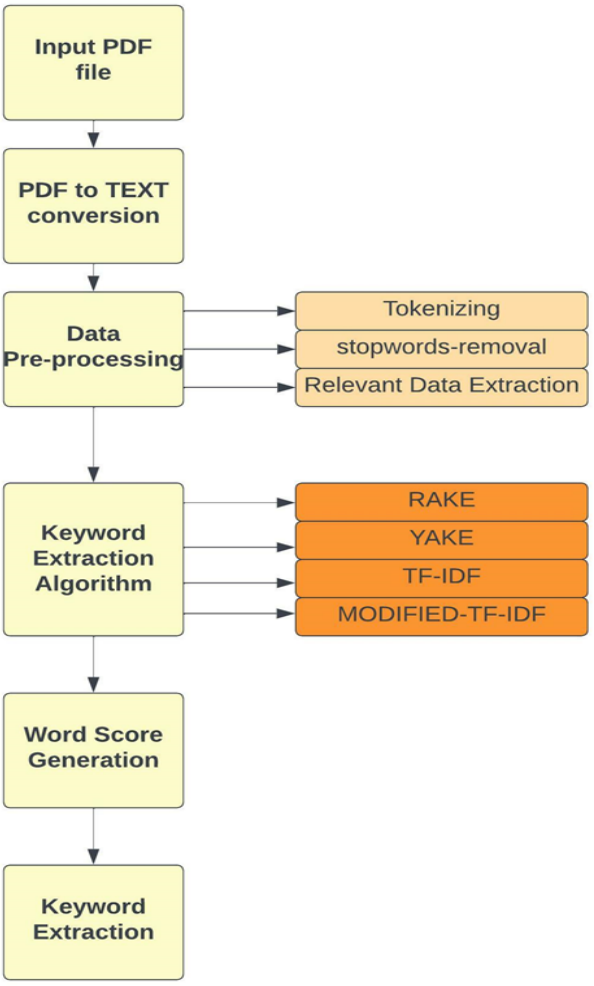
Keyword extraction helps you sift through the whole set of data and obtain the words that best describe each review in just seconds. That way, you can easily and automatically see what words are mentioned most often, saving your hours upon hours of manual processing.

The main purpose of the project is to take PDF input from users and extract meaningful keywords from the text. Different algorithms are used to process pdf files and conversion of files into text and then extraction of keywords.Our approach provides the result in a minimum time span with maximum precision and accuracy in comparison to other existing approaches.

1.4 BRIEF DESCRIPTION OF THE SOLUTION APPROACH:

We have used a guided approach in studying and implementing the various models of opinion. First we have taken the pdf file as a input after that we have converted the pdf file into the text and after conversion of text we have done the data pre-processing in that we have done the Tokenization , Removing of stop-words and Relevant Data Extraction after that we have implemented different types of Keyword Extraction Algorithm i.e Rake, Yake, Tf-Idf and modified them as per our need to get best result .

Our approach involved a lot of research to fully understand the given model and then converting to code for the given model. On the completion of the code, the results were extracted.



CHAPTER 2

LITERATURE SURVEY

2.1) SUMMARY OF PAPERS STUDIED

We studied more than 30 different research papers related to the topic **Keyword Extraction.**
In our attempt to find a model that can be translated into code for evaluating the results, we went through 10 different research papers each to find different models for study. Through our research, we decided to study and try to implement RAKE, YAKE,TF-IDF. We also studied more research papers to find similar models.

2.2) INTEGRATED SUMMARY OF THE PAPERS STUDIED:

1) Rapid Automatic Keyword Extraction and Word Frequency in Scientific Article Keywords Extraction

PUBLISHED IN: - 2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS)

DATE OF PUBLICATION: - 28 December 2021

SUMMARY: - In this research paper both word frequency and RAKE algorithm are combined to get more accurate in extracting keywords from an article.

2) Keyword Extraction from Scientific Research Projects Based on SRP-TF-IDF

PUBLISHED IN - Chinese Journal of Electronics

DATE OF PUBLICATION: - 1 July 2021

SUMMARY: - In this article a new approach of keyword extraction is implemented i.e., SRP-TF-IDF model in which the normal TF-IDF is combined with the weight balance algorithm designed to recalculate candidate keywords.

3) Improving the Keyword Co-occurrence Analysis: An Integrated Semantic Similarity Approach

PUBLISHED IN - 2021 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)

DATE OF PUBLICATION: - 19 January 2022

SUMMARY: - In this research paper RAKE algorithm and semantic similarity are combined together and analyzed the unweighted and weighted matrices in terms of their network structure and cluster quality.

4) Extraction of Trend Keywords from Thai Twitters using N-Gram Word Combination

PUBLISHED IN - 2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)

DATE OF PUBLICATION: - 04 August 2020

SUMMARY: - This paper proposes a method for extracting keywords from Thai text on social media. A N-gram-based word-combination technique is presented to segment words that are not in dictionaries and increase the precision of word segmentation.

5) Automatic Keywords Extraction Based on Co-Occurrence and Semantic Relationships between Words

PUBLISHED IN - IEEE

DATE OF PUBLICATION: - 24 June 2020

SUMMARY :- This paper, propose an unsupervised keywords extraction framework for individual documents, which improves the keywords extraction from two aspects i.e., word co-occurrence and semantic relationships and use Precision, Recall, and F1-measure values as evaluation criteria to compare all keywords extraction methods proposed with other strong baseline methods in two datasets.

CHAPTER 3

REQUIREMENT ANALYSIS AND SOLUTION APPROACH

3.1 Hardware and Software dependency and prerequisites

3.1.1. Hardware Used

· Processor: Intel i3 10gen 64 bit or Ryzen 3 or higher

· RAM: 4 GB

· Hard disk Space: 10 GB

· An Internet Connection

3.1.2. Software Used

· Operating System: Windows, Linux or Mac

· Code Editor: PyCharm IDE

3.1.3. Libraries Used:

· Pandas
· PyPDF2
· nltk
· math
· orderedset
· yake

3.1.4. Language Used

· Python

3.2 Solution Approach:

Based on Module 1, the work related to Conversion of PDF to TEXT has been discussed below:

PDF to TEXT using PyPDF2 module:

PyPDF2 is a free and open source pure-python PDF library used for performing fundamental tasks on PDF files including extracting the report-unique facts, merging the PDF files, splitting the pages of a PDF document, adding watermarks to a document, encrypting and decrypting the PDF documents, etc. we have used the PyPDF2 library in this project because It miles a natural python library so it can run on any platform with none platform-related dependencies on any outside libraries.

Based on Module 2, We have Implemented Keyword Extraction Algorithms which are RAKE,YAKE,TF-IDF.

CHAPTER 4

MODELING AND IMPLEMENTATION DETAILS

4.1 MODELING

4.1.1 RAKE: In this minor project, firstly we have studied the RAKE Technique [2][3]. RAKE also known as Rapid Keyword Extraction Algorithm

The concept behind this algorithm is to make 3 matrices namely word degree word frequency and degree to frequency ratio and calculating score based on them:

- Word degree matrix simply means that the no of times a given word co-occurs with another word.

	feature	extraction	complex	algorithm	available	help	rapid	automatic	keyword
feature	2	2	0	0	0	0	0	0	0
extraction	2	3	0	0	0	0	0	1	1
complex	0	0	1	0	0	0	0	0	0
algorithm	0	0	0	1	1	0	0	0	0
available	0	0	0	1	1	0	0	0	0
help	0	0	0	0	0	1	0	0	0
rapid	0	1	0	0	0	0	1	1	1
automatic	0	1	0	0	0	0	1	1	1
keyword	0	1	0	0	0	0	1	1	1
TOTAL SUM	**4**	**8**	**1**	**2**	**2**	**1**	**3**	**4**	**4**

Candidate Key Phrases :
- ✓ feature extraction
- ✓ Complex
- ✓ help
- ✓ algorithms available
- ✓ rapid automatic keyword extraction

Total **Unique** candidate key phrases count is 5. but total 'Feature' is coming with 2 candidate key phrases . Hence 2

- Word frequency matrix basically means how many times a single word reappears in the given phrase or a para.

Word	Degree Of word	Word Frequency	Degree Score
feature	4	2	2
extraction	8	3	2.66
complex	1	1	1
algorithm	2	1	2
available	2	1	2
help	1	1	1
rapid	3	1	3
automatic	4	1	4
keyword	4	1	4

- Degree to frequency ratio matrix is the ratio of word degree matrix and word frequency matrix as follows

$$Deg_{to_{freq}} = \frac{Word\ degree\ matrix}{word\ frequency\ matrix}$$

Cumulative Score of candidate Key Phrases				
feature	extraction			4.66
Complex				1
algorithms	available			4
help				1
rapid	automatic	keyword	extraction	13.66

4.1.2 YAKE: We have also studied the YAKE[4][5] which is unsupervised approach also known as Yet Another Keyword Extraction which can be used for multiple languages and simply extract keyword on the basis of some parameters which are mentioned below

1. PRE-PROCESSING AND CANDIDATE GENERATION: remove punctuation marks and stop words.

2. CANDIDATE SCORING:

a. CASING:

Check for capitalized letter word and acronym form.

$$casing(w) = \frac{max(count(w\ is\ capital), count(w\ is\ acronym))}{1+log(count(w))} \qquad (a)$$

b. WORD POSITIONAL:

First, we get position of all sentences where it finds the word "w".

$$sen(w) = position\ of\ sentences\ where\ w\ occurs$$

Then, Position Feature is calculated using formula:

$$position(w) = log(\log log\ (3\ +\ Median(sen(w))))\ ) \qquad (b)$$

c. WORD FREQUENCY:

It tells us the frequency of words

$$frequency(w) = \frac{count\ of\ word\ w}{mean(counts)+standard\ deviation(counts)} \qquad (c)$$

d. WORD RELATEDNESS TO CONTENT:

This feature tells us how much the word is related to the context.

$$relatedness(w) = 1\ +\ (WR\ +\ WL)\ *\ \frac{count(w)}{count}\ +\ PL\ +\ PR \qquad (d)$$

WR = (number of unique words on right) / (total words on right)

WL = (number of unique words on left) / (total words on left)

PL = (total words on left) / (max count)

PR = (total words on right) / (max count)

e. WORD DIFFERENT SENTENCE:

It tells us how often a word appear in different sentences.

$$different(w) = \frac{number\ of\ sentences\ w\ occurs\ in}{total\ sentences} \qquad (e)$$

COMBINE WORD SCORE:-

It scores the single word using the formula

$$score(w) = \frac{d*b}{a+(c/d)+(e/d)}$$

KEYWORD SCORE:-

For each candidate keyword, a score calculated using formula:

$$S(kw) = \frac{product(scores\ of\ words\ in\ keyword)}{1+(sum\ of\ scores\ of\ words)*count(keyword)}$$

3. POST PROCESSING:-

It removes the word with the similar meanings or duplicates using Levenshtein distance.

For e.g.:

“work”, “works” ,

“relevant”, “relevance”

4.FINAL RANKING:

The final list of keywords is generated with their scores lower the score more important the keyword.

4.1.3 TF-IDF: Tf-idf is one of the best metrics for evaluating a term's importance to a corpus or series of texts. A weighting technique called tf-idf gives each word in a document a weight depending on the frequency of its terms (tf) and the reciprocal frequency of the document (tf) (idf). The words with higher weight scores are thought to be more important .

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

Document Frequency: This evaluates the meaning of the text in the entire corpus collection that is quite similar to TF. The sole distinction is that in document d, TF represents the frequency counter for a phrase t, whereas df represents the quantity of times the term t appears in the document set N. In other words, there are DF papers that contain the word.

df(t) = Document occurrence of t

The primary purpose of Inverse Document Frequency is to evaluate a word's relevance. Finding the pertinent records that meet the demand is the search's primary goal.

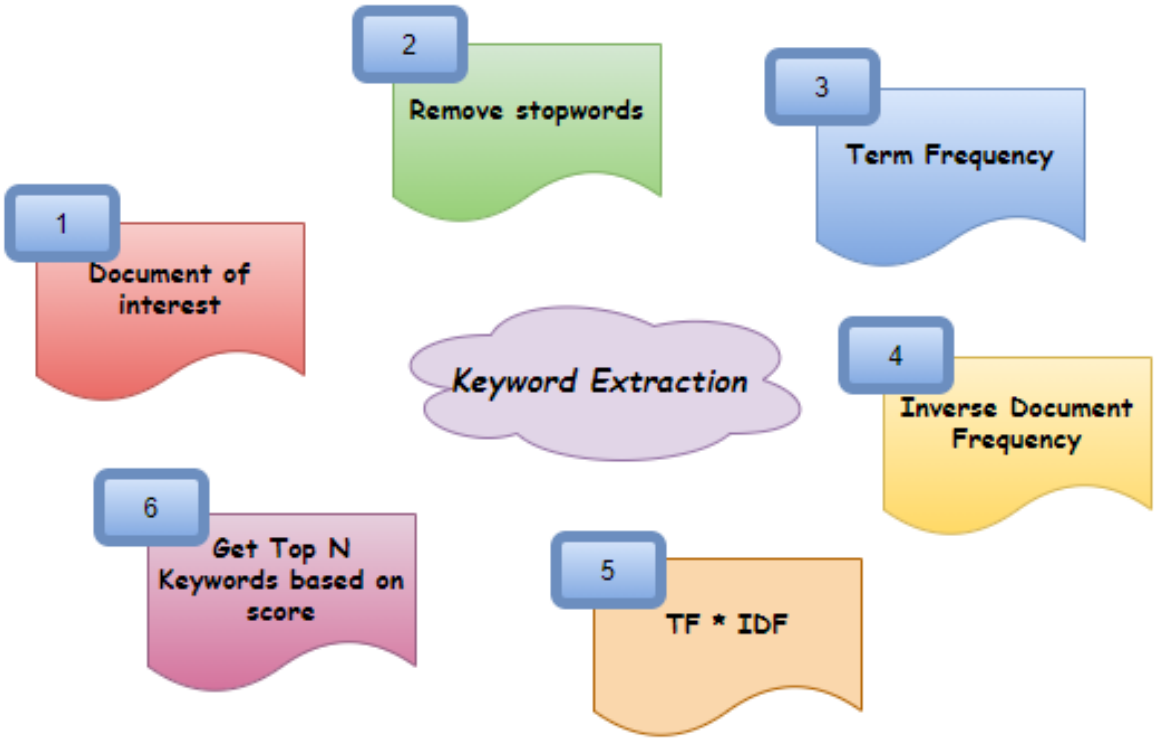
df(t) = N(t) where df(t) equals the term's t's document frequency

N(t) = Number of records that contain the letter t

The number of documents in the corpus divided by the text's frequency makes up the word's IDF.

IDF (t) = N/DF (t) = N/N (t)

N/ df(t) = log(idf(t)



4.1.4 MODIFIED TF-IDF:

Normal tf-idf is for normal documents but we have modified it for the research papers in the way that it doesn't consider already mentioned keywords and references in the research paper which makes our algorithm more efficient and give more efficient keywords.

4.2 IMPLEMENTATION:

1)PDF-TO-TEXT:

```
# importing required modules
import PyPDF2
# enter name of input and output file
filename = input("enter name of input pdf file: ")
outputfilen = input("enter output file with extension: ")
# opening pdf file
pdfFileObj = open(filename, 'rb')
# creating a pdf reader object
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
# printing number of pages in pdf file
n = pdfReader.numPages
#print(n)

for i in range(n):
    page = pdfReader.getPage(i)
    text = page.extractText()
    file1 = open(outputfilen, 'a', errors = 'ignore')
    print(text)
    file1.writelines(text)
    file1.close()
pdfFileObj.close()
```

2)RAKE:

```
import operator
import os
import PyPDF2
import re
# Utility function to load stop words from a file and return as a list of words
def loadStopWords(stopWordFile):
    stopWords = []
    for line in open(stopWordFile):
        if (line.strip()[0:1] != "#"):
            for word in line.split():  # in case more than one per line
                stopWords.append(word)
    return stopWords
# Utility function to return a list of all words that are have a length greater than a specified number of characters.
def separatewords(text, minWordReturnSize):
    splitter = re.compile('[^a-zA-Z0-9_\\+\\-/]')
    words = []
    for singleWord in splitter.split(text):
        currWord = singleWord.strip().lower()
        # leave numbers in phrase, but don't count as words, since they tend to invlate scores of their phrases
        if len(currWord) > minWordReturnSize:
            words.append(currWord)
    return words
# Utility function to return a list of sentences.
def splitSentences(text):
    sentenceDelimiters = re.compile(u'[.!?,;:\t\\-\\"\\(\\)\\'\u2019\u2013]')
    sentenceList = sentenceDelimiters.split(text)
    return sentenceList
def buildStopwordRegExPattern(pathtostopwordsfile):
    stopwordlist = loadStopWords(pathtostopwordsfile)
    stopwordregexlist = []
```

```
    for wrd in stopwordlist:
        wrdregex = '\\b' + wrd + '\\b'
        stopwordregexlist.append(wrdregex)
    stopwordpattern = re.compile('|'.join(stopwordregexlist), re.IGNORECASE)
    return stopwordpattern
def generateCandidateKeywords(sentenceList, stopwordpattern):
    phraseList = []
    for s in sentenceList:
        tmp = re.sub(stopwordpattern, '|', s.strip())
        phrases = tmp.split("|")
        for phrase in phrases:
            phrase = phrase.strip().lower()
            if (phrase != ""):
                phraseList.append(phrase)
    return phraseList
def calculateWordScores(phraseList):
    wordfreq = {}
    worddegree = {}
    for phrase in phraseList:
        wordlist = separatewords(phrase, 0)
        wordlistlength = len(wordlist)
        wordlistdegree = wordlistlength - 1
        # if wordlistdegree > 3: wordlistdegree = 3 #exp.
        for word in wordlist:
            wordfreq.setdefault(word, 0)
            wordfreq[word] += 1
            worddegree.setdefault(word, 0)
            worddegree[word] += wordlistdegree
    for item in wordfreq:
        worddegree[item] = worddegree[item] + wordfreq[item]
```

```
    wordscore = {}
    for item in wordfreq:
        wordscore.setdefault(item, 0)
        wordscore[item] = worddegree[item] / (wordfreq[item] * 1.0)
    return wordscore
def generateCandidateKeywordScores(phraseList, wordscore):
    keywordcandidates = {}
    for phrase in phraseList:
        keywordcandidates.setdefault(phrase, 0)
        wordlist = separatewords(phrase, 0)
        candidatescore = 0
        for word in wordlist:
            candidatescore += wordscore[word]
        keywordcandidates[phrase] = candidatescore
    return keywordcandidates
def rake(text):
    sentenceList = splitSentences(text)
    stoppath = os.path.join(os.path.dirname(__file__), "SmartStoplist.txt")
    stopwordpattern = buildStopwordRegExPattern(stoppath)
    # generate candidate keywords
    phraseList = generateCandidateKeywords(sentenceList, stopwordpattern)
    # calculate individual word scores
    wordscores = calculateWordScores(phraseList)
    # generate candidate keyword scores
    keywordcandidates = generateCandidateKeywordScores(phraseList, wordscores)
    sortedKeywords = sorted(keywordcandidates.items(), key=operator.itemgetter(1), reverse=True)
    print(sortedKeywords[5:])
    return sortedKeywords
```

3)YAKE:

```
import yake
import PyPDF2
import re
import nltk
from nltk.tokenize import word_tokenize

filename = input("enter name of input pdf file: ")
# opening pdf file
pdfFileObj = open(filename, 'rb')
# creating a pdf reader object
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
# printing number of pages in pdf file
n = pdfReader.numPages
for i in range(n):
    page = pdfReader.getPage(i)
    text = page.extractText()
pdfFileObj.close()
text = text.lower()  # convert all words to lower case,

text = re.sub(r'\s+', ' ', text)  # replace or substitute all spaces, tabs, indents (\s) with ' ' (space)

text = re.sub(r'\d', ' ', text)  # replace all digits by ' '
```

```
split_string = text.split("-", 1)
text = split_string[0]
filtered_list = []
stop_words = nltk.corpus.stopwords.words('english')

    # Tokenize the sentence
words = word_tokenize(text)
for w in words:
    if w.lower() not in stop_words:
        filtered_list.append(w)

    text = " ".join(filtered_list)
language = "en"
max_ngram_size = 1
numOfKeywords = 20

kw_extractor = yake.KeywordExtractor(lan=language,
                                     n=max_ngram_size,
                                     top=numOfKeywords)

keywords = kw_extractor.extract_keywords(text)

for kw in keywords:
    print(kw)
```

4)TF-IDF:

```
def get_sentence_of_words(text):
    sentence = list()  # list of sentences
    words = list()  # list of words in each sentence.

    sentence_list = list()

    temp = text.strip().split(". ")  # temporary list of sentences.

    for sent in temp:

        words = sent.strip().split(" ")  # getting the words in sentences.

        words = [i for i in words if len(i) > 1]

        if (len(words) > 1):
            sentence.append(words)  # sentence is a list of lists.contains a list of sentences in which each sentence
            # is a list of words

        sentence_list.append(sent)

    #print(sentence,print(len(sentence))
    return sentence, sentence_list
```

```python
def vectorize(sentence):
    # set of unique words in the whole document.
    unique_words = OrderedSet()
    for sent in sentence:
        for word in sent:
            unique_words.add(word)
    unique_words = list(unique_words)  # converting the set to a list to make it easier to work with it.

    #print(unique_words, len(unique_words))
    # a list of lists that contains the vectorized form of each sentence in the document.
    vector = list()
    for sent in sentence:  # iterate for every sentence in the document
        temp_vector = [0] * len(
            unique_words)  # create a temporary vector to calculate the occurence of each word in that sentence.
        for word in sent:  # iterate for every word in the sentence.
            temp_vector[unique_words.index(word)] += 1
        vector.append(temp_vector)  # add the temporary vector to the list of vectors for each sentence (list of lists)
    #print(vector)
    return vector, unique_words
```

```python
def tf(vector, sentence, unique_words):
    tf = list()

    no_of_unique_words = len(unique_words)

    for i in range(len(sentence)):

        tflist = list()
        sent = sentence[i]
        count = vector[i]

        for word in sent:
            score = count[sent.index(word)] / float(len(sent))  # tf = no. of occurence of a word/ total no. of words

            if (score == 0):
                score = 1 / float(len(sentence))

            tflist.append(score)

        tf.append(tflist)

    #print(tf)

    return tf
```

```
def idf(vector, sentence, unique_words):
    # idf = log(no. of sentences / no. of sentences in which the word appears).

    no_of_sentences = len(sentence)

    idf = list()

    for sent in sentence:

        idflist = list()

        for word in sent:

            count = 0  # no. of times the word occurs in the entire text.

            for k in sentence:
                if (word in k):
                    count += 1

            score = math.log(no_of_sentences / float(count))  # caclulating idf scores

            idflist.append(score)

        idf.append(idflist)

    #print(idf)

    return idf
```

```
def tf_idf(tf, idf):
    # tf-idf = tf(w) * idf(w)

    tfidf = [[0 for j in range(len(tf[i]))] for i in range(len(tf))]

    for i in range(len(tf)):
        for j in range(len(tf[i])):
            tfidf[i][j] = tf[i][j] * float(idf[i][j])

    # print(tfidf)

    return tfidf
```

```
def extract_keywords(tfidf, processed_text):
    mapping = {}
    for i in range(len(tfidf)):
        for j in range(len(tfidf[i])):
            mapping[processed_text[i][j]] = tfidf[i][j]
    mapping = dict(sorted(mapping.items(), key=lambda value: value[1]))
    print(mapping)
    word_scores = sorted(mapping.values(), reverse=True)
    words = []
    scores_to_word = {}
    for i in range(len(tfidf)):
        for j in range(len(tfidf[i])):
            scores_to_word[tfidf[i][j]] = processed_text[i][j]

    for i in range(len(word_scores)):
        if (word_scores[i] != 0):
            words.append(scores_to_word[word_scores[i]])
        else:
            words.append(scores_to_word[word_scores[i]])
            break
    #print(words)
    words = OrderedSet(words)
    for i in mapping:
        if (mapping[i] == 0):
            words.append(i)
    #print(words, mapping)
    return words, mapping
```

```
def save_keywords(words, mapping):
    scores = []
    for word in words:
        scores.append(mapping[word])
    # print(words, scores)
    d = {'Words': words, 'Scores': scores}
    data = pd.DataFrame(d)
    data.to_csv('keywords.csv', sep='\t')
    print(data)
```

5)MODIFIED TF-IDF:

```
    text = text.replace("keyword","[")
    text = text.replace("introduction","]")
    text = "".join(re.split("[\[\]]", text)[::2])
    text = text.replace("reference", "-")
    split_string = text.split("-", 1)
    text = split_string[0]
```

CHAPTER 5

TESTING

5.1 Testing Plan:

Testing is a crucial element of software quality assurance and presents ultimate review of specification, design and coding. System testing is an important phase. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

Testing is planned with 5 different research papers.

5.2 Testing Objectives:

1. Testing is a process of executing a program with the intent of finding an error.

2. A good test case is one that has a probability of finding an as yet undiscovered error

3. A successful test is one that uncovers an undiscovered error.

5.3 Output:

1)DOCUMENT 1:

RAKE:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/rake.py
enter name of input pdf file: test1.pdf
[('zimmermann key vec automatic ranked keyphrase extraction', 41.133333333333326),
('pollak rakun rankbased  extraction information processing  management', 39.75),
 ('mccallum semeval      task    scienceie  extracting keyphrases', 36.0),
('computational linguistics human language technologies association', 26.833333333333332),
('hulth improved automatic extractiondatasets thesis', 25.333333333333332),
 ('multiple local features information sciences', 24.25),
('daille topicrank graphbased topic ranking', 24.0),
('baldwin automatic keyphrase extraction', 17.133333333333333),
('natural language processing      pp', 17.083333333333332),
('martin sgrank combining statistical', 16.0),
('high utility pattern clustering', 16.0),
('arabic documents information systems', 15.25),
('sixth international joint conference', 14.166666666666666),
('knowledge management      pp', 11.25),
('unsupervised keyphrase extraction', 10.8),
('keyphrase extraction system', 10.8),
('computational linguistics      pp', 10.416666666666666),
('wang topic detection', 10.0),
('computational semantics association', 9.833333333333334),
('articles language resources', 9.333333333333332),
('orleans louisiana      pp', 9.25),
('fourth joint conference', 9.166666666666666),
('north american chapter', 9.0),
('keyphrase extraction', 7.8)]
```

YAKE:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/yakeee.py
enter name of input pdf file: test1.pdf
('extraction', 0.06985791569571284)
('computational', 0.0726314315472919)
('keyphrase', 0.09662750287200741)
('association', 0.11458636255219543)
('linguistics', 0.11458636255219543)
('scienti', 0.15127245592286792)
('conference', 0.152179803607389)
('information', 0.15402482605032508)
('url', 0.1576600127527403)
('proceedings', 0.165483624312758)
('doi', 0.17223248004233863)
('automatic', 0.19468725482037808)
('language', 0.19468725482037808)
('danesh', 0.2188455295124351)
('sumner', 0.2188455295124351)
('martin', 0.2188455295124351)
('sgrank', 0.2188455295124351)
('combining', 0.2188455295124351)
('statistical', 0.2188455295124351)
('graphical', 0.2188455295124351)
```

TF-IDF:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/main.py
enter name of input pdf file: test1.pdf
          Words    Scores
0       digital  0.240864
1            th  0.192691
2      sciences  0.183102
3       systems  0.160576
4         given  0.152125
5       natural  0.146482
6     gregation  0.144519
7     semantics  0.125668
8     documents  0.122068
9           url  0.099542
10    louisiana  0.096346
11  linguistics  0.094303
12  information  0.083560
13           pp  0.073241
14     language  0.071163
15    keyphrase  0.061034
16   extraction  0.038508
```

MODIFIED:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/modified.py
enter name of input pdf file: test1.pdf
          Words    Scores
0       systems  0.197304
1       natural  0.170997
2     documents  0.143985
3   linguistics  0.124787
4           url  0.112795
5     semantics  0.111520
6    conference  0.097756
7   information  0.090666
8     louisiana  0.085498
9            pp  0.078577
10    keyphrase  0.073501
11   extraction  0.047618
```

DOCUMENT 2:

RAKE:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/rake.py
enter name of input pdf file: test2.pdf
[('nevillmanning kea practical auto matic keyphrase extraction', 42.666666666666664),
 ('local word vectors guiding keyphrase extraction', 27.666666666666668),
 ('manchester uk      coling      companion volumeposters', 25.0),
 ('discovering citation intent classification', 16.0),
 ('deep recurrent neural networks', 16.0),
 ('single document keyphrase extraction', 15.666666666666668),
 ('exploiting contextual word embedding', 15.5),
 ('natural language processing emnlp', 14.666666666666666),
 ('information processing  management           doi', 14.0),
 ('natural language processing', 10.666666666666666),
 ('glove global vectors', 10.5),
 ('word representa tion', 9.5),
 ('latent dirichlet allocation', 9.0),
 ('paukkeri ms nieminen', 9.0),
 ('expert systems applications', 9.0),
 ('peerj computer science', 8.5),
 ('keyphrase extraction', 7.666666666666667),
 ('fourth acm conference', 7.25),
 ('automatic  extraction', 5.5),
 ('york acm', 4.5),
 ('peerj comput', 4.5),
 ('complexity             doi', 4.333333333333334),
 ('sentence clustering', 4.0),
 ('international workshop', 4.0),
 ('languageindependent approach', 4.0),
 ('computational linguistics', 4.0),
 ('manning cd', 4.0)]
```

YAKE:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/yakeee.py
enter name of input pdf file: test2.pdf
('extraction', 0.06207596880555238)
('keyphrase', 0.08950308631092632)
('proceedings', 0.09969230760365848)
('conference', 0.16329223817745103)
('word', 0.17372188704433245)
('doi', 0.19859605261871896)
('papagiannopoulou', 0.20495304823187027)
('processing', 0.220109403357934)
('tsoumakas', 0.24142199413142518)
('wang', 0.2435926660675996)
('single', 0.29518419870065554)
('evaluation', 0.3052716026846888)
('document', 0.32009737915904163)
('empirical', 0.3302293395811933)
('methods', 0.3302293395811933)
('natural', 0.3302293395811933)
('language', 0.3302293395811933)
('peerj', 0.3343256745288546)
('vectors', 0.33782972798824634)
('international', 0.34650606222516706)
```

TF-IDF:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/main.py
enter name of input pdf file: test2.pdf
          Words    Scores
0     tsoumakas  1.925074
1    representa  0.641691
2       twitter  0.481268
3      external  0.427794
4          word  0.410642
5        latent  0.385015
6     embedding  0.350013
7        comput  0.081918
8         peerj  0.067170
9    processing  0.058543
10        using  0.052422
11   conference  0.047675
12  proceedings  0.043795
13    keyphrase  0.040516
14   extraction  0.030899
```

MODIFIED:

```
A:\pythonProject2\venv\Scripts\python.exe A:/pythonProject2/modified.py
enter name of input pdf file: test2.pdf
          Words    Scores
0     tsoumakas  1.763180
1            co  0.705272
2    representa  0.587727
3     knowledge  0.503766
4       twitter  0.440795
5        latent  0.352636
6     embedding  0.320578
7        comput  0.103716
8         peerj  0.083330
9    processing  0.071404
10   conference  0.062943
11  proceedings  0.056380
12    keyphrase  0.051018
13   extraction  0.042556
```

CHAPTER 6:

CONCLUSION FUTURE WORK AND REFERENCES

CONCLUSION:

We can easily convert a pdf file to text and can extract meaningful keywords from it.We have presented different state-of-art techniques used in the field of extracting keywords with a code implementation for each of them. Each of the four methods has its own advantages. Each of them succeeded in extracting keywords that are either identical to the keywords specified by the author or close to them and related to the field. We have studied many different types of algorithms and finally worked on Rake, Yake ,Tf-Idf and Modified them to get the best Keywords as our result. However, we are successful at Extraction of Keywords which are best matching with the document but in comparison we conclude that the Tf-Idf algorithm is best suitable for extracting keywords from documents.

FUTURE WORK:

In future we will try to modify the current algorithm which we have implemented so to get the better results also we will try to implement new algorithm through which we can extract keywords from text document such as term weighting scheme which is based on the traditional TF-IDF and uses a cluster of synonyms from thesaurus of the respective domain which will helps in the consideration of the words which are synonyms of each other, thus making use of the semantic similarity between the words.

Currently we are extracting keywords from the text document but in future we will try to extract information from the images as well which are there in the document to get the better results as well as some new approach can be used that eliminates the noisy data which could further reduce the computational time of the algorithm.

REFERENCES:

[1]https://towardsdatascience.com/keyword-extraction-a-benchmark-of-7-algorithms-in-python-8a905326d93f

[2] https://doi.org/10.1002/9780470689646.ch1

[3]https://www.analyticsvidhya.com/blog/2021/10/rapid-keyword-extraction-rake-algorithm-in-natural-language-processing

[4] https://doi.org/10.1016/j.ins.2019.09.013

[5]https://repositorio.inesctec.pt/server/api/core/bitstreams/ef121a01-a0a6-4be8-945d-3324a58fc944/content

[6] https://aclanthology.org/C16-2015.pdf