AI and ML Techniques for Cyber Security

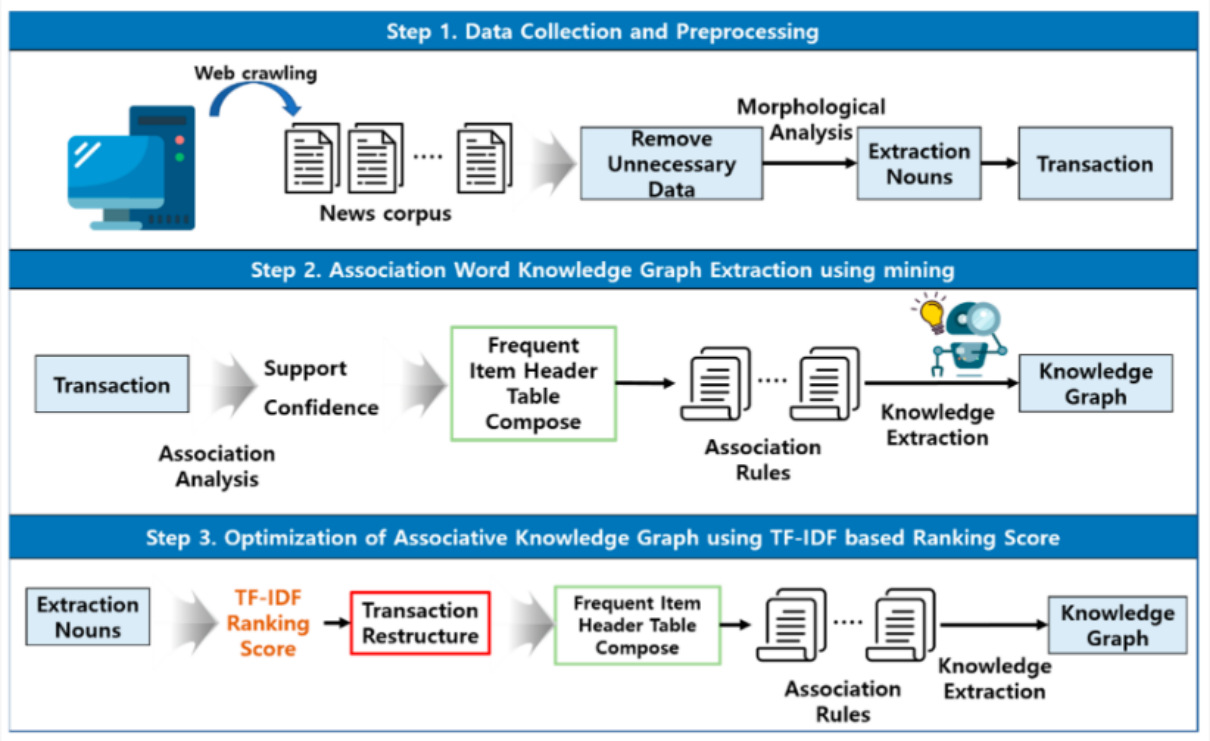
Assignment 1: Text Summarization using Python & NLTK: TF-IDF Algorithm

Contents

Objective:

A TF-IDF-based text summarization model can automatically extract and highlight the most relevant sentences from a given document. The model aims to leverage Term Frequency-Inverse Document Frequency (TF-IDF) to identify key sentences that best represent the core content of the text. By scoring sentences based on the significance of their words, the model will generate concise summaries that effectively capture the main ideas while minimizing redundancy. This approach is intended to provide a quick and efficient means of distilling important information from large volumes of text.

Figure 4. Optimizing process of association graph using TF-IDF ranking score.



1. Overall process description & solution approach

1.1 The text summarization solution described in the provided file involves several key steps:

1. **Sentence Tokenization**: The text is split into individual sentences using the `sent_tokenize` function from the NLTK library. This forms the basis for analysing each sentence separately.

2. **Frequency Matrix Creation**: For each sentence, a frequency matrix is created that tracks the frequency of each word. The words are first converted to lowercase and stemmed to their root form using the PorterStemmer. Stop words (common words that don't contribute much to the meaning, like "and" or "the") are removed.

3. **Term Frequency (TF) Calculation**: The term frequency for each word in a sentence is calculated by dividing the frequency of the word by the total number of words in the sentence. This results in a matrix where each word has a term frequency score.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

4. **Documents per Words Table Creation**: A table is generated that keeps track of how many sentences (documents) each word appears in. This is useful for calculating the IDF (Inverse Document Frequency).

5. **Inverse Document Frequency (IDF) Calculation**: The IDF for each word is calculated using the formula:

IDF(t) = log_10(Total number of documents / Number of documents with term)

This measures how unique or rare a word is across all sentences.

6. **TF-IDF Matrix Calculation**: The TF and IDF values are multiplied to generate a TF-IDF matrix. This matrix helps in identifying the importance of words in the context of the entire text.

7. **Sentence Scoring**: Each sentence is scored by summing up the TF-IDF values of the words it contains. The score reflects the overall importance of the sentence within the text.

8. **Threshold Calculation**: An average score is calculated from all sentence scores. This threshold helps to determine which sentences are important enough to be included in the summary.

9. **Summary Generation**: Sentences that have a score above a certain threshold (1.3 times the average score) are selected and concatenated to form the final summary.

This approach leverages the TF-IDF technique to identify and extract the most significant sentences, creating a concise summary of the original text.

1.2 Process diagram:

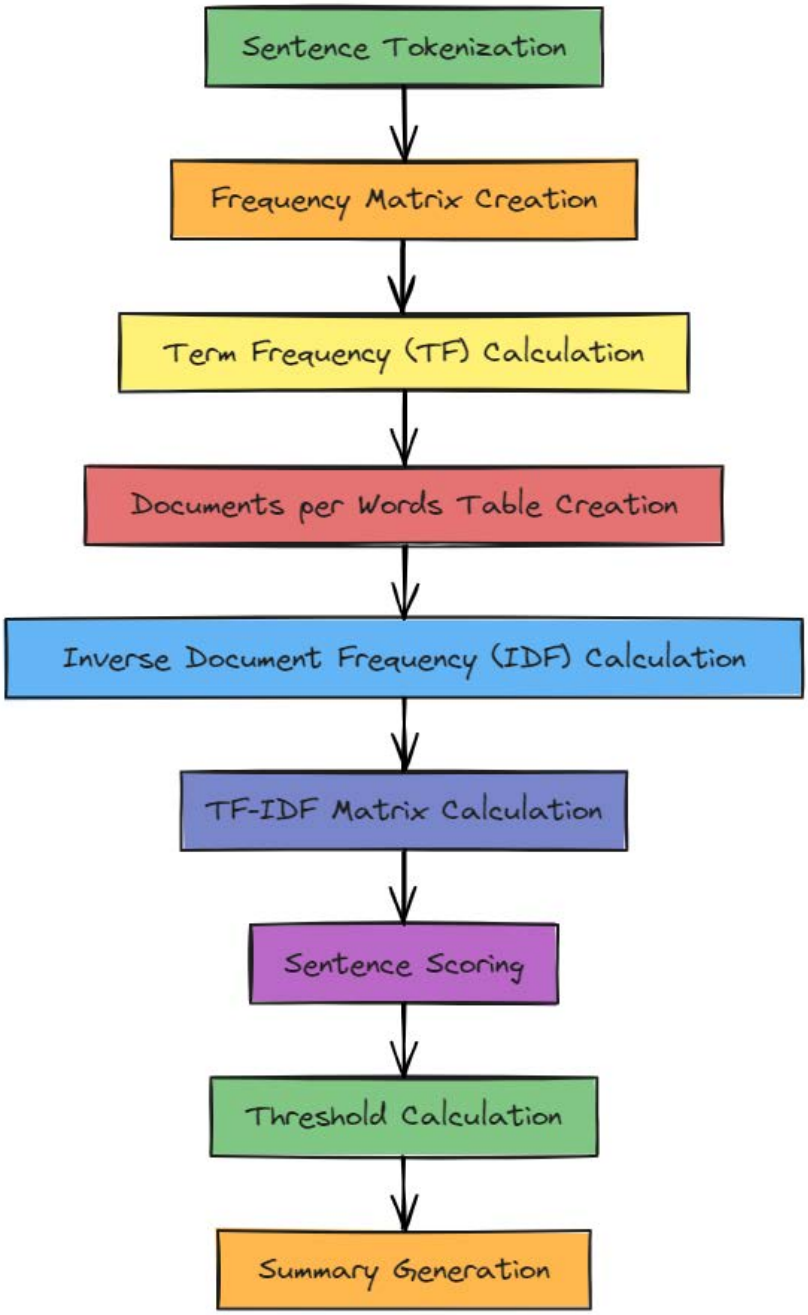


Figure 1: process diagram

2. Tool used and reasons to use this specific tool

2.1 Tools used

Name	Description
Python	The programming language to implement the algorithm.
NLTK (Natural Language Toolkit)	For sentence and word tokenization, stopwords removal, and stemming.

2.2 Reasons to use above tools

Name	Reason
Python	**Ease of Use** Python's syntax is clear and readable. **Extensive Libraries** There are many libraries available for natural language processing and machine learning.
NLTK (Natural Language Toolkit)	**Comprehensive Toolkit** It provides easy-to-use interfaces to over 50 corpora and lexical resources. **Preprocessing Capabilities** Includes tools for text preprocessing, such as tokenization, stemming, lemmatization, and stopword removal.

3. Source code snippets

Imports

```
import math

import nltk
nltk.download('punkt')
nltk.download('stopwords')

from nltk import sent_tokenize, word_tokenize, PorterStemmer
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
pip install pdfplumber
```

```
Requirement already satisfied: pdfplumber in /usr/local/lib/python3.10/dist-packages (0.11.4)
Requirement already satisfied: pdfminer.six==20231228 in /usr/local/lib/python3.10/dist-packages (from pdfplumber) (20231228)
Requirement already satisfied: Pillow>=9.1 in /usr/local/lib/python3.10/dist-packages (from pdfplumber) (9.4.0)
Requirement already satisfied: pypdfium2>=4.18.0 in /usr/local/lib/python3.10/dist-packages (from pdfplumber) (4.30.0)
Requirement already satisfied: charset-normalizer>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from pdfminer.six==20231228->pdfpl
Requirement already satisfied: cryptography>=36.0.0 in /usr/local/lib/python3.10/dist-packages (from pdfminer.six==20231228->pdfplumber
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography>=36.0.0->pdfminer.six==20231228
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography>=36.0.0->pdfminer.si
```

Read the pdf file

```
#text = "Those Who Are Resilient Stay In The Game Longer. On the mountains of truth you can never climb in vain: either you will reach a

import pdfplumber

# Path to the PDF file
pdf_file_path = '/content/Paytm secures finance ministry approval for investment in payment services business - Times of India.pdf'

# Open the PDF file
with pdfplumber.open(pdf_file_path) as pdf:
    # Initialize a variable to store the extracted text
    text = ''

    # Iterate through the pages and extract text
    for page in pdf.pages:
        text += page.extract_text()

# Display the extracted text
print(text)

```

```
Printed from
Paytm secures finance ministry approval for
investment in payment services business
TIMESOFINDIA.COM | Aug 28, 2024, 08.46 PM IST
Paytm, a prominent fintech company, has secured approval from
the finance ministry to invest in its payment services business, the
company announced on Wednesday.
The company, officially known as One 97 Communications, has
been under the watchful eye of banking regulator and financial
crime-fighting agency following the central bank's directive in
January to shut down its payments bank.
With the recent approval, Paytm plans to submit a new
application to the ministry to regain the license for its payments
services business. The company said that Paytm Payment
Services will continue to provide online payment aggregation
services to existing partners.
Although Paytm did not disclose the specifics of the approved investment, Reuters reported in July, based on
information from a senior finance ministry official, that the company had obtained approval for a 500 million
rupee (approximately $6 million) investment in its payments division.
Paytm Payment Services, a significant component of the fintech firm's operations, contributed a quarter of its
consolidated revenue in the fiscal year that ended in March 2023.
In July, Vivek Joshi, financial services secretary, said that the company could approach India's central bank toapply for a payment agg
On the day of the announcement, Paytm's shares closed 1.3% lower. Since the central bank's order to wind down
the payment bank in January, the company's shares have declined by more than 29%.
```

1. Tokenize the sentences

```
sentences = sent_tokenize(text) # NLTK function
total_documents = len(sentences)
```

2. Create the Frequency matrix of the words in each sentence.

```
def _create_frequency_matrix(sentences):
    frequency_matrix = {}
    stopWords = set(stopwords.words("english"))
    ps = PorterStemmer()

    for sent in sentences:
        freq_table = {}
        words = word_tokenize(sent)
        for word in words:
            word = word.lower()
            word = ps.stem(word)
            if word in stopWords:
                continue

            if word in freq_table:
                freq_table[word] += 1
            else:
                freq_table[word] = 1

        frequency_matrix[sent[:15]] = freq_table

    return frequency_matrix
```

3. Calculate TermFrequency and generate a matrix

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

```
def _create_tf_matrix(freq_matrix):
    tf_matrix = {}

    for sent, f_table in freq_matrix.items():
        tf_table = {}

        count_words_in_sentence = len(f_table)
        for word, count in f_table.items():
            tf_table[word] = count / count_words_in_sentence

        tf_matrix[sent] = tf_table

    return tf_matrix
```

4. Creating a table for documents per words

```
def _create_documents_per_words(freq_matrix):
    word_per_doc_table = {}

    for sent, f_table in freq_matrix.items():
        for word, count in f_table.items():
            if word in word_per_doc_table:
                word_per_doc_table[word] += 1
            else:
                word_per_doc_table[word] = 1

    return word_per_doc_table
```

5. Calculate IDF and generate a matrix

IDF(t) = log_e(Total number of documents / Number of documents with term t in it)

```
def _create_idf_matrix(freq_matrix, count_doc_per_words, total_documents):
    idf_matrix = {}

    for sent, f_table in freq_matrix.items():
        idf_table = {}

        for word in f_table.keys():
            idf_table[word] = math.log10(total_documents / float(count_doc_per_words[word]))

     idf_matrix[sent] = idf_table

```

6. Calculate TF-IDF and generate a matrix

TF-IDF algorithm is made of 2 algorithms multiplied together.

```
def _create_tf_idf_matrix(tf_matrix, idf_matrix):
    tf_idf_matrix = {}

    for (sent1, f_table1), (sent2, f_table2) in zip(tf_matrix.items(), idf_matrix.items()):

        tf_idf_table = {}

        for (word1, value1), (word2, value2) in zip(f_table1.items(),
                                                    f_table2.items()):  # here, keys are the same in both the table
            tf_idf_table[word1] = float(value1 * value2)

        tf_idf_matrix[sent1] = tf_idf_table

    return tf_idf_matrix
```

7. Score the sentences

```
def _score_sentences(tf_idf_matrix) -> dict:
    """
    score a sentence by its word's TF
    Basic algorithm: adding the TF frequency of every non-stop word in a sentence divided by total no of words in a sentence.
    :rtype: dict
    """

    sentenceValue = {}

    for sent, f_table in tf_idf_matrix.items():
        total_score_per_sentence = 0

        count_words_in_sentence = len(f_table)
        for word, score in f_table.items():
            total_score_per_sentence += score

        sentenceValue[sent] = total_score_per_sentence / count_words_in_sentence

    return sentenceValue
```

8. Find the threshold

```
def _find_average_score(sentenceValue) -> int:
    """
    Find the average score from the sentence value dictionary
    :rtype: int
    """
    sumValues = 0
    for entry in sentenceValue:
        sumValues += sentenceValue[entry]

    # Average value of a sentence from original summary_text
    average = (sumValues / len(sentenceValue))

    return average
```

9. Generate the summary

```
def _generate_summary(sentences, sentenceValue, threshold):
    sentence_count = 0
    summary = ''

    for sentence in sentences:
        if sentence[:15] in sentenceValue and sentenceValue[sentence[:15]] >= (threshold):
            summary += " " + sentence
            sentence_count += 1

    return summary
```

Execution of the text summerization

```
import math

from nltk import sent_tokenize, word_tokenize, PorterStemmer
from nltk.corpus import stopwords

'''
We already have a sentence tokenizer, so we just need
to run the sent_tokenize() method to create the array of sentences.
'''
# 1 Sentence Tokenize
sentences = sent_tokenize(text)
total_documents = len(sentences)
#print(sentences)

# 2 Create the Frequency matrix of the words in each sentence.
freq_matrix = _create_frequency_matrix(sentences)
#print(freq_matrix)

'''
Term frequency (TF) is how often a word appears in a document, divided by how many words are there in a document.
'''
# 3 Calculate TermFrequency and generate a matrix
tf_matrix = _create_tf_matrix(freq_matrix)
#print(tf_matrix)

# 4 creating table for documents per words
count_doc_per_words = _create_documents_per_words(freq_matrix)
#print(count_doc_per_words)

'''
Inverse document frequency (IDF) is how unique or rare a word is.
'''
# 5 Calculate IDF and generate a matrix
idf_matrix = _create_idf_matrix(freq_matrix, count_doc_per_words, total_documents)
#print(idf_matrix)

# 6 Calculate TF-IDF and generate a matrix
tf_idf_matrix = _create_tf_idf_matrix(tf_matrix, idf_matrix)
#print(tf_idf_matrix)

# 7 Important Algorithm: score the sentences
sentence_scores = _score_sentences(tf_idf_matrix)
#print(sentence_scores)

# 8 Find the threshold
threshold = _find_average_score(sentence_scores)
#print(threshold)

# 9 Important Algorithm: Generate the summary
summary = _generate_summary(sentences, sentence_scores, 1.3 * threshold)
print(summary)
```

```
 The company said that Paytm Payment
Services will continue to provide online payment aggregation
services to existing partners. On the day of the announcement, Paytm's shares closed 1.3% lower.
```

4. Final output results and analysis of results

4.1 Final Output Results:

The final output of the text summarization process is a concise summary of the input text. This summary is generated by selecting the most significant sentences based on their TF-IDF scores. The sentences that were scored above a certain threshold were included in the summary.

4.2 Analysis of Results:

- Efficiency: The summarization process efficiently reduces the text size while retaining the most important information. The use of TF-IDF helps to prioritize sentences that contain key concepts.
- Quality: The quality of the summary is dependent on the accuracy of the TF-IDF calculations. The method effectively identifies and retains sentences with high information density.
- Limitations: The summarization is purely extractive, meaning it only selects and compiles existing sentences. It does not rephrase or condense ideas, which could sometimes lead to a summary that is less fluid or coherent.

Overall, the approach provides a straightforward and effective way to generate summaries, especially useful for large texts where quick comprehension of key points is needed.