

# Text-Summarizer

Autorzy:

*W. Zamarski*

*F. Tryhuk*

*S. Zych*

# Spis treści

1. Abstrakt.....	2
2. Wstęp.....	2
2.1 Cel.....	2
2.2 Zakres.....	2
2.3 Metodyka.....	3
3. Część Teoretyczna.....	3
4. Część Praktyczna.....	3
4.1 Wczytanie tekstu źródłowego.....	3
4.2 Stworzenie worka słów.....	4
4.3 Wyliczenie średnich wartości zdań.....	4
4.4 Generowanie podsumowania tekstu.....	5
4.5 GUI.....	6
5. Podsumowanie.....	6
6. Bibliografia.....	6
Dodatek - Kod GUI.....	6

## 1. Abstrakt

Generowanie automatycznych podsumowań tekstu jest jednym z istotnych zastosowań metod przetwarzania języków naturalnych. Do tego problemu podejść można na wiele sposobów, ale najpopularniejsze są rozwiązania ekstrakcyjne i abstrakcyjne. Chociaż metody abstrakcyjne cieszą się ostatnio rosnącą popularnością, ich wysoki poziom skomplikowania czyni je trudnymi w implementacji oraz często obarcza je ciężkimi do wykrycia błędami. Z tych powodów, w naszym projekcie skupiliśmy się na podejściu ekstrakcyjnym, umożliwiającym błyskawiczne uzyskanie najistotniejszych informacji z tekstów o znacznej długości.

## 2. Wstęp

### 2.1 Cel

Celem naszego projektu było napisanie programu pozwalającego na automatyczne tworzenie podsumowań tekstów metodą ekstrakcyjną.

### 2.2 Zakres

Nasz program obsługuje fragmenty tekstów w języku angielskim. Zawierać powinien różne możliwości wstawiania tekstu do podsumowania, np. otwarcie go z pliku .txt, pobranie tekstu z strony internetowej poprzez podanie jej url, ręczne wpisanie tekstu (w tym możliwość użycia skrótów "ctrl+c", "ctrl+v"). Ponadto użytkownik powinien mieć możliwość wybrania długości

podsumowania poprzez możliwość podania jak ważne powinno być zdanie względem reszty tekstu aby zostało ujęte w efekcie końcowym. Po przekazaniu tekstu oraz podaniu minimalnej istotności zdań, tekst powinien zostać automatycznie podsumowany.

## 2.3 Metodyka

Do pisania programu użyty został język Python oraz poniższe jego biblioteki:

- tkinter - stworzenie interfejsu graficznego,
- nltk - przetwarzanie tekstu,
- re - czyszczenie tekstu,
- bs4 - parsowanie zawartości strony internetowej,
- urllib - pobieranie zawartości strony internetowej,

Na początku pracy nad projektem skupiliśmy się na uzyskaniu działającego szkieletu programu streszczającego tekst. W tym celu przeszukaliśmy istniejące rozwiązania i wybraliśmy to, które okazało się najbardziej odpowiadające naszym wymaganiom. Następnie rozbudowaliśmy znaleziony program o funkcjonalności, które uznaliśmy za najpotrzebniejsze, np. wczytanie tekstu z pliku. Po dodaniu ich, program – będący do tej pory aplikacją działającą przez CLI – obudowany został w łatwy w obsłudze interfejs graficzny.

## 3. Część Teoretyczna

Aby można było zastosować narzędzia do przetwarzania języków naturalnych, wczytany tekst musi zostać podzielony i przetworzony na części składowe – stokenizowane zdania. Tekst dzielony jest na zdania za pomocą PunktSentenceTokenizer.

Przetworzony w ten sposób tekst można podać do przetwarzania, mającemu na celu uzyskanie tekstu "czystszej". Na tym etapie można zastosować wiele różnych metod mających na celu poprawienie analizy tekstu. W rozwiązaniu użytym przez nas ograniczamy się do odfiltrowania słów stopu (stopwords) oraz pozbycia się znaków pozbawionych znaczenia lub o znaczeniu niejasnym dla analizy, takich jak interpunkcja albo cyfry.

Następnie dokonywana jest właściwa analiza zdań. Zdania – a właściwie tablice słów odpowiadających poszczególnym zdaniom – są sortowane i zliczane w celu wyznaczenia tokenów (słów) występujących najczęściej, ponieważ słowa takie zawierają zazwyczaj istotną treść którą chcemy otrzymać w końcowym streszczeniu. Wszystkim słowom są następnie przypisywane wagi odpowiadające częstotliwości ich występowania, a następnie suma tych wag jest dodawana osobno dla każdego zdania by wyznaczyć kluczowe fragmenty tekstu – zdania zawierające więcej "ważnych" słów.

Po wyznaczeniu średniej wagi przeciętnego zdania, porównywane są z nią sumy wag wszystkich wag. Jeśli waga danego zdania jest większa od średniej pomnożonej przez ustaloną przez użytkownika wartość progową (domyślnie 1.5), to zdanie wybierane jest do tworzonego streszczenia. Po przeiterowaniu po całym tekście, zwracane jest podsumowanie składające się ze zdań o wybranej przez użytkownika minimalnej istotności.

## 4. Część Praktyczna

W projekcie wykorzystano zmodyfikowany kod ze źródła [1]. Ponadto, wprowadzono klasę GUI, której zadaniem jest zapewnienie wygodnej obsługi programu poprzez interfejs graficzny.

### 4.1 Wczytanie tekstu źródłowego

Istnieje możliwość pobrania tekstu ze strony internetowej, poprzez wyodrębnienie treści z tagów <p> lub wczytania tekstu z pliku.

```
def _read_from_webpage(webpage):
    #fetching the content from the URL
    fetched_data = urllib.request.urlopen(webpage)
    article_read = fetched_data.read()
    #parsing the URL content and storing in a variable
    article_parsed = BeautifulSoup.BeautifulSoup(article_read, 'html.parser')
    #returning <p> tags
    paragraphs = article_parsed.find_all('p')
    article_content = ''
    #looping through the paragraphs and adding them to the variable
    for p in paragraphs:
        s = str(p.text)
        s = re.sub('\[\d+\]', '', s)
        article_content += s
    return article_content

def _read_from_file(file_name):
    file = open(file_name, "r")
    filedata = file.readlines()
    return filedata[0]
```

### 4.2 Stworzenie worka słów

Wykorzystano stemmer Portera i słowa stopu z pakietu nltk.corpus. Wynikiem funkcji jest słownik reprezentujący częstość występowania poszczególnych słów.

```
def _create_dictionary_table(text_string) -> dict:
    nltk.download('stopwords')
    nltk.download('punkt')
    #removing stop words
    stop_words = set(stopwords.words("english"))
    words = word_tokenize(text_string)
    #reducing words to their root form
    stem = PorterStemmer()

    #creating dictionary for the word frequency table
    frequency_table = dict()
    for wd in words:
        if wd in stop_words:
            continue
```

```

wd = stem.stem(wd)
if wd in frequency_table:
    frequency_table[wd] += 1
else:
    frequency_table[wd] = 1

return frequency_table

```

### 4.3 Wyliczenie średnich wartości zdań

Oceniane jest każde zdanie, poprzez przypisaniu mu wagi na podstawie sumy wag słów w nim zawartych.

```

def _calculate_sentence_scores(sentences, frequency_table) -> dict:
    #algorithm for scoring a sentence by its words
    sentence_weight = dict()

    for sentence in sentences:
        sentence_wordcount_without_stop_words = 0
        for word_weight in frequency_table:
            if word_weight in sentence.lower():
                sentence_wordcount_without_stop_words += 1
            if sentence[:7] in sentence_weight:
                sentence_weight[sentence[:7]] += frequency_table[word_weight]
            else:
                sentence_weight[sentence[:7]] = frequency_table[word_weight]
        sentence_weight[sentence[:7]] = sentence_weight[sentence[:7]] /
        sentence_wordcount_without_stop_words

    return sentence_weight

```

Obliczenie średniej oceny zdań umożliwia ustalenie progu istotności dla generowanego podsumowania.

```

def _calculate_average_score(sentence_weight) -> int:
    #calculating the average score for the sentences
    sum_values = 0
    for entry in sentence_weight:
        sum_values += sentence_weight[entry]

    #getting sentence average value from source text
    average_score = (sum_values / len(sentence_weight))

    return average_score

```

### 4.4 Generowanie podsumowania tekstu

Proces ten polega na wybieraniu zdań o ocenach powyżej ustalonego progu (threshold), tworząc w ten sposób podsumowanie. Jeśli zdania mają zbliżony stopień istotności jest

możliwe uzyskanie pustego tekstu na wyjściu, dlatego proces jest powtarzany z obniżonym progiem istotności.

```
def _get_article_summary(sentences, sentence_weight, threshold):
    sentence_counter = 0
    article_summary = ''

    while not article_summary:
        for sentence in sentences:
            if sentence[:7] in sentence_weight and sentence_weight[sentence[:7]] >=
(threshold):
                article_summary += " " + sentence
                sentence_counter += 1

        #If article_summary is still empty, lower the threshold and try again
        threshold -= 0.1
        if threshold <= 0.2:
            break

    return article_summary
```

Cała funkcjonalność programu została umieszczona w funkcji `_run_article_summary`, co umożliwia łatwe uruchamianie pełnego procesu.

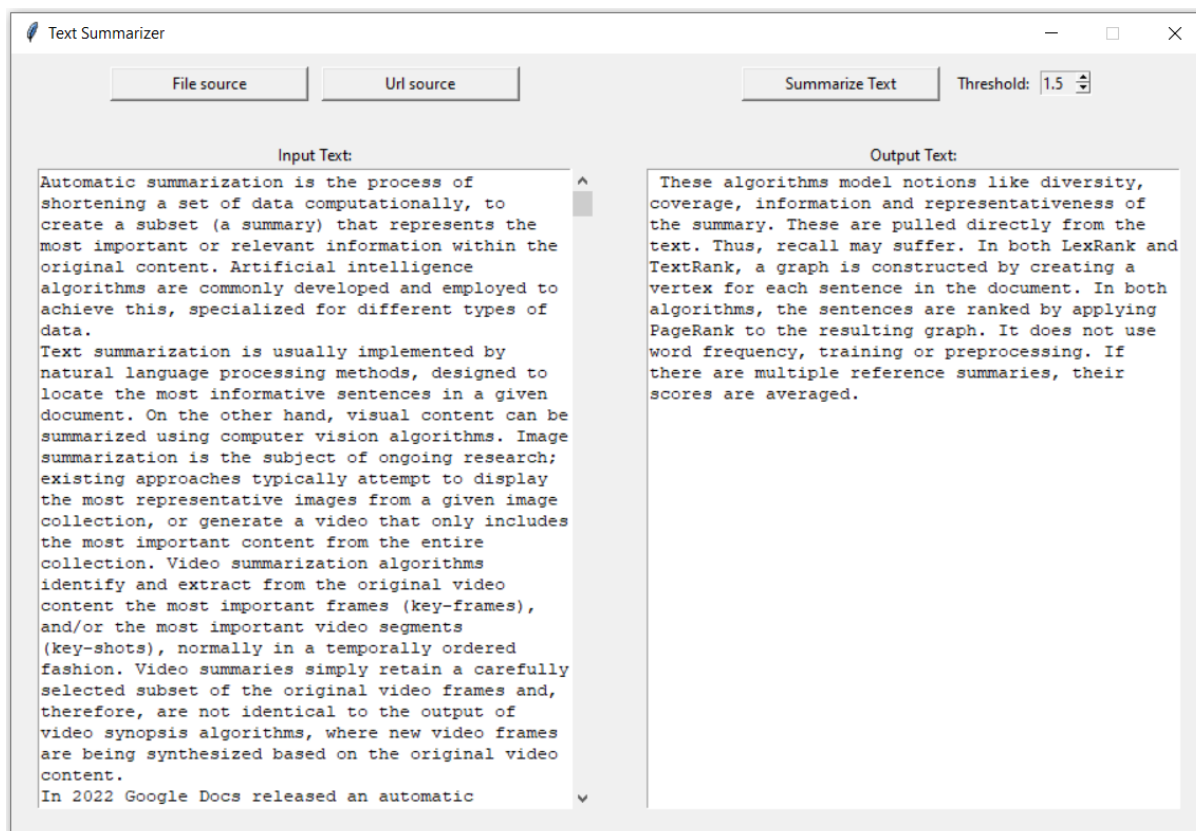
```
def _run_article_summary(article, treshold_multiplier = 1.5):
    #creating a dictionary for the word frequency table
    frequency_table = _create_dictionary_table(article)
    #tokenizing the sentences
    sentences = sent_tokenize(article)
    #algorithm for scoring a sentence by its words
    sentence_scores = _calculate_sentence_scores(sentences, frequency_table)
    #getting the threshold
    threshold = _calculate_average_score(sentence_scores) * treshold_multiplier
    #producing the summary
    article_summary = _get_article_summary(sentences, sentence_scores, threshold)

    return article_summary
```

## 4.5 GUI

Kod klasy GUI umieszczono w dodatku. Interfejs graficzny składa się z trzech części:

- panelu kontrolnego - zawiera przyciski do kontroli programu
- tekstu źródłowego - zawiera tekst do podsumowania. Można uzupełnić to pole ręcznie lub użyć opcji wczytania tekstu z pliku lub strony internetowej.
- tekstu wyjściowego - zawiera podsumowany tekst (bez możliwości edycji)



Rys. 1. Interfejs graficzny

## 5. Podsumowanie

Utworzono rozbudowaną o interfejs graficzny aplikację, która przyjmuje od użytkownika tekst wczytany z pliku lub z adresu strony internetowej, następnie przetwarza go i zwraca streszczenie o poziomie szczegółowości wybranym przez użytkownika.

## 6. Bibliografia

- [1] [blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/](http://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/)
- [2] [github.com/edubey/text-summarizer](https://github.com/edubey/text-summarizer)

## Dodatek - Kod GUI

```
import tkinter as tk
import summarization
from tkinter import scrolledtext, filedialog, simpledialog

class GUI:
    def __init__(self):
        root = tk.Tk()
```

```

root.title("Text Summarizer")
root.resizable(False, False)

source_frame = tk.Frame(root, padx=20, pady=10)
control_frame = tk.Frame(root, padx=20, pady=10)
input_frame = tk.Frame(root, padx=20, pady=20)
output_frame = tk.Frame(root, padx=20, pady=20)

source_frame.grid(row=0, column=0)
control_frame.grid(row=0, column=1)
input_frame.grid(row=1, column=0)
output_frame.grid(row=1, column=1)

tk.Button(source_frame, text="File source", width=20,
command=self.use_file_source).grid(
    row=0, column=0, padx=5)
tk.Button(source_frame, text="Url source", width=20,
command=self.use_url_source).grid(
    row=0, column=1, padx=5)

tk.Button(control_frame, text="Summarize Text", width=20,
command=self.summarize_text).grid(
    row=0, column=0, padx=5)
tk.Label(control_frame, text="Threshold:").grid(row=0, column=1, padx=5)

self._threshold_input = tk.Spinbox(control_frame, from_=1.0, to=2.0, increment=0.1,
state='readonly', width=4,
                                textvariable=tk.DoubleVar(value=1.5))
self._threshold_input.grid(row=0, column=2)

tk.Label(input_frame, text="Input Text:").grid(row=0, column=0)
tk.Label(output_frame, text="Output Text:").grid(row=0, column=0)

self._text_input = scrolledtext.ScrolledText(input_frame, wrap=tk.WORD, width=50,
height=30)
self._text_output = tk.Text(output_frame, wrap=tk.WORD, state=tk.DISABLED, width=50,
height=30)
self._text_input.grid(row=1, column=0)
self._text_output.grid(row=1, column=0)

root.mainloop()

def use_file_source(self):
    file_path = tk.filedialog.askopenfilename(filetypes=[("*.txt files", "*.txt")])
    if file_path:
        self.set_input_text(summarization._read_from_file(file_path))

def use_url_source(self):
    url = tk.simpledialog.askstring("Url source", "Enter URL:")
    if url:
        self.set_input_text(summarization._read_from_webpage(url))

def summarize_text(self):

```



```

        self.set_output_text(summarization._run_article_summary(self.get_input(),
float(self.get_threshold())))

def set_input_text(self, text):
    self._text_input.delete(1.0, tk.END)
    self._text_input.insert(tk.END, text)
    self.set_output_text("")

def set_output_text(self, text):
    self._text_output.config(state=tk.NORMAL)
    self._text_output.delete(1.0, tk.END)
    self._text_output.insert(tk.END, text)
    self._text_output.config(state=tk.DISABLED)

def get_threshold(self):
    return self._threshold_input.get()

def get_input(self):
    return self._text_input.get("1.0", 'end-1c')

if __name__ == "__main__":
    gui = GUI()

```