Narzędzie do tworzenia podsumowań tekstu

ClipSum

Autorzy:

Michał Lechowicz

Damian Kuciel

Wojciech Najdek

Abstrakt

Praca ta ma na celu zwiększenie zrozumienia i świadomości na temat podsumowywania tekstu, a w szczególności skupia się na porównaniu dwóch podejść do tego procesu: ekstrakcyjnego i abstrakcyjnego.

W części teoretycznej przedstawiono definicje obu podejść oraz omówiono ich zalety i wady. Zwrócono uwagę na to, że podejście ekstrakcyjne jest prostsze i skuteczniejsze, ale może nie uwzględniać pewnych istotnych elementów tekstu. Z kolei podejście abstrakcyjne jest bardziej skomplikowane, ponieważ zmaga się z takimi problemami, jak interpretacja dokumentu i generowanie języka naturalnego.

Z kolei w części praktycznej przedstawiono projekt narzędzia do tworzenia podsumowań tekstu, który opiera się na podejściu ekstrakcyjnym. Omówiono etapy realizacji projektu oraz możliwe rozszerzenia narzędzia, takie jak dodanie funkcji tłumaczenia na inne języki.

Wstęp

Cel projektu

Celem projektu jest stworzenie narzędzia, które dla podanego tekstu wejściowego zwróci jego podsumowanie. W tym przypadku będzie to aplikacja internetowa streszczająca artykuły z kilku popularnych serwisów informacyjnych. Dodatkowo użytkownik będzie miał możliwość załadowania swojego tekstu, który ma zostać skrócony. Część projektu, która odpowiedzialna jest za skracanie tekstu, zostanie oparta na repozytorium *text-summarizer* użytkownika *edubey*.

Opis problemu

Streszczanie tekstu (text summarization) to jedno z zagadnień przetwarzania języka naturalnego i polega ono, jak sama nazwa wskazuje, na skracaniu tekstu źródłowego i wyciąganiu z niego najważniejszych informacji przy zachowaniu jego ogólnego sensu.

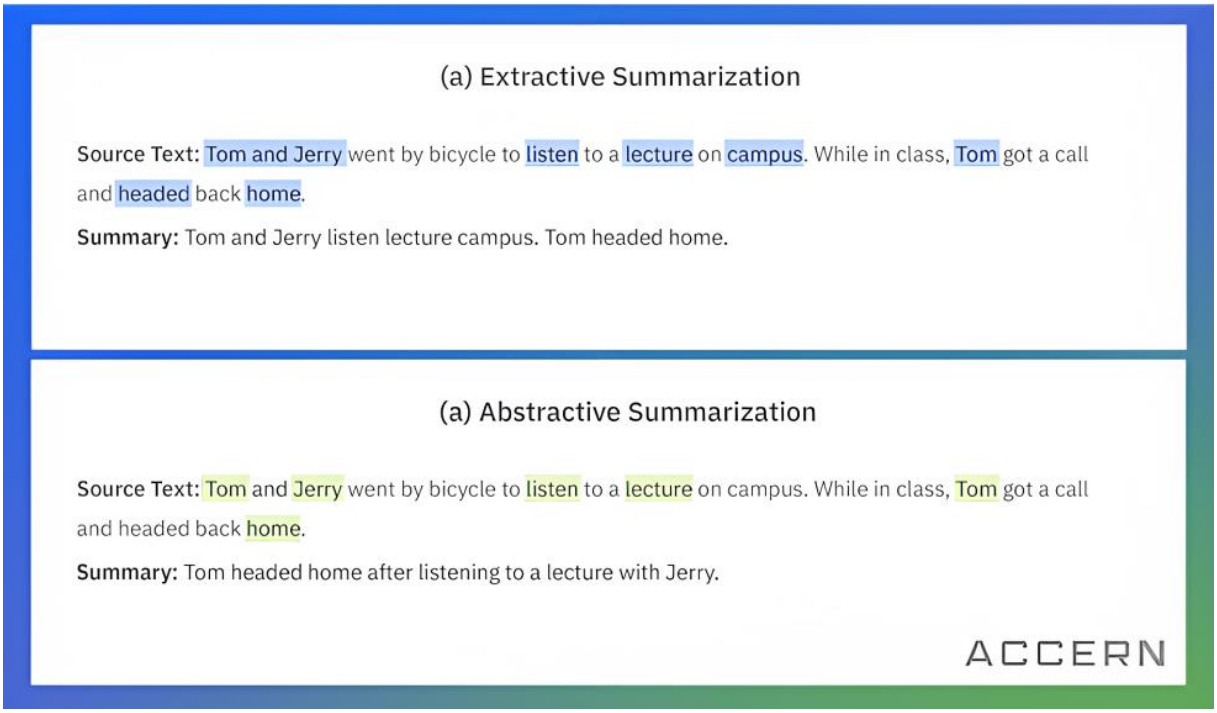
Podsumowywanie tekstu może okazać się przydatne w wielu sytuacjach, np.:

- codzienne skracanie artykułów ze stron internetowych interesujących użytkownika
- podsumowanie badań naukowych
- podsumowanie książek
- podsumowanie raportów biznesowych

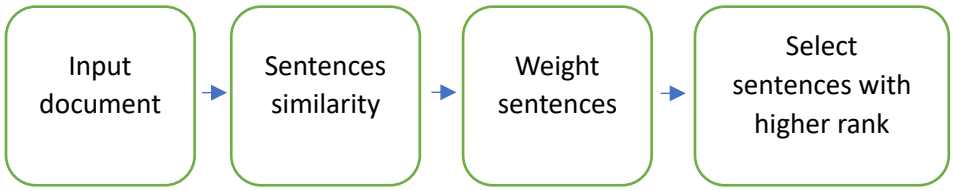
Występują dwa podejścia do streszczania tekstów: ekstrakcyjne (extractive) i abstrakcyjne/abstraktowe (abstractive).
Podejście ekstrakcyjne polega na „wyciągnięciu" z tekstu źródłowego najważniejszych fragmentów i umieszczenie ich w podsumowaniu. Podejście to może zwracać lepsze rezultaty (streszczenia) od podejścia abstrakcyjnego, ponieważ nie występuje tutaj problem dotyczący interpretacji tekstu wejściowego.
Natomiast podejście abstrakcyjne tworzy streszczenie z wykorzystaniem zaawansowanych technik przetwarzania języka naturalnego. Konieczne jest tutaj skorzystanie z technik pozwalających na zinterpretowanie dokumentu i generowanie języka naturalnego, a w związku z tym proces streszczania tekstu jest bardziej skomplikowany niż w podejściu ekstrakcyjnym. Zwrócony tekst może zawierać słowa

nie występujące w tekście oryginalnym. Podejście abstrakcyjne bardziej przypomina sposób, w jaki człowiek streściłby dany tekst.

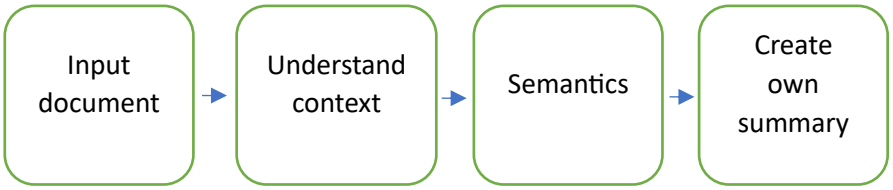
Przykład obrazujący różnicę pomiędzy tymi podejściami:



Podejście ekstrakcyjne można przedstawić na rysunku w taki sposób:



Podejście abstrakcyjne prezentuje się w taki sposób:



Na podstawie powyższych grafik można zauważyć, że kroki które należy podjąć w podejściu abstrakcyjnym są bardziej skomplikowane.

Rozwinięcie

Implementacja podejścia ekstrakcyjnego

Ze względu na swoją prostotę i skuteczność, do streszczania artykułów w projekcie, zostało zaimplementowane podejście ekstrakcyjne.

Algorytm wykorzystywany w projekcie został napisany w języku Python.

Wykorzystane biblioteki:

- **nltk** – (Natural Language Toolkit) – biblioteka zawierająca zestaw modułów i funkcji do przetwarzania języka naturalnego.
- **numpy** – biblioteka ułatwiająca programistom pracę z wektorami, macierzami oraz wielowymiarowymi tablicami. Biblioteka ta wprowadza m.in. swój typ danych nazwany ndarray, który jest wydajniejszy od listy dostępnej w języku Python.
- **networkx** – biblioteka pozwalająca na pracę z grafami i sieciami w Pythonie.

Funkcja służąca do generacji podsumowań:

```
def generate_summary(text, top_n=5, is_file_path=False, min_length=128):
    stop_words = stopwords.words('english')
    summarize_text = []

    # Step 1 - Read text and split it
    sentences = read_article(text, is_file_path)

    # Step 2 - Generate Similarity Matrix across sentences
    sentence_similarity_matrix = build_similarity_matrix(sentences, stop_words)

    # Step 3 - Rank sentences in similarity matrix
    sentence_similarity_graph = nx.from_numpy_array(sentence_similarity_matrix)
    scores = nx.pagerank(sentence_similarity_graph)

    # Step 4 - Sort the rank and pick top sentences
    ranked_sentence = sorted(((scores[i], s) for i, s in enumerate(sentences)),
reverse=True)
    # print("Indexes of top ranked_sentence order are ", ranked_sentence)

    if len(ranked_sentence) < top_n:
        top_n = len(ranked_sentence)

    for i in range(top_n):
        if len("".join(summarize_text)) >= min_length:
            break
        summarize_text.append(" ".join(ranked_sentence[i][1]))
        summarize_text.append(". ")

    # Step 5 - output the summarized text
    return "".join(summarize_text)
```

Funkcję *generate_summary* można podzielić na pięć kroków:

Krok 1. Algorytm wczytuje tekst, który ma być podsumowany, a następnie dzieli go na zdania.

Krok 2. Algorytm tworzy macierz podobieństwa między każdą parą zdań, wykorzystując funkcję *sentence_similarity*. Macierz ta jest wypełniana wartościami liczbowymi reprezentującymi podobieństwo między każdą parą zdań.

Krok 3. Na podstawie utworzonej macierzy podobieństwa, algorytm tworzy graf reprezentujący związki między zdaniami, a następnie wykorzystuje algorytm PageRank do obliczenia rankingów dla każdego zdania.

Krok 4. Algorytm sortuje zdania na podstawie wyników otrzymanych w poprzednim kroku i wybiera top_n zdań,które mają być użyte w streszczeniu.

Krok 5. Algorytm łączy wybrane zdania w jedno streszczenie i zwraca je jako wynik działania

Opis algorytmu PageRank

PageRank został opracowany w 1996 roku przez założycieli Google - Larry'ego Page'a i Sergeya Brina. Pierwotnie został opracowany jako algorytm oceny istotności stron internetowych w sieci WWW. Nazwa "PageRank" została utworzona od nazwiska Larry'ego Page'a i od słowa "rank", które odnosi się do pozycji strony w wynikach wyszukiwania. Algorytm PageRank wykorzystuje graf skierowany, w którym wierzchołkami są strony internetowe, a krawędzie reprezentują hiperłącza między nimi. Algorytm analizuje strukturę sieci i oblicza wartości PageRank dla każdej strony internetowej. Wartość ta odpowiada szacowanej istotności strony i jest wykorzystywana do ustawienia pozycji strony w wynikach wyszukiwania. Pierwotny patent wygasł w 2018 roku. Google nadal używa algorytmu do pozycjonowania stron w swojej wyszukiwarce.

W kontekście rozpatrywanego kodu, PageRank jest wykorzystywany do oceny istotności każdego zdania w dokumencie na podstawie grafu podobieństwa między zdaniami. Najpierw tworzona jest macierz podobieństwa między zdaniem a zdaniem, a następnie tworzony jest graf, którego wierzchołkami są zdania, a krawędziami są wagi odpowiadające podobieństwu między nimi. PageRank jest używany do obliczenia wartości dla każdego wierzchołka w grafie, co odpowiada jego istotności. Zdania są następnie sortowane według tych wartości i wybierane są najlepsze zdania jako podsumowanie. W ten sposób, algorytm PageRank pomaga w automatycznym wykrywaniu istotnych fragmentów tekstu poprzez określenie, które zdania najlepiej reprezentują treść całego dokumentu.

Funkcja pomocnicza służąca do wczytywania tekstu:

Funkcja *read_article* odpowiadająca za wczytanie artykułu, podzielenie go na zdania oraz oczyszczenie zdań ze zbędnych znaków:

```
def read_article(input, is_file_path):
    if is_file_path:
        file = open(input, "r")
        filedata = file.readlines()
        article = filedata[0].split(". ")
    else:
        article = input.split(". ")
    sentences = []

    for sentence in article:
        # print(sentence)
        sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
    sentences.pop()

    return sentences
```

Funkcja odpowiedzialna za obliczanie podobieństwa między zdaniami:

```
def sentence_similarity(sent1, sent2, stopwords=None):
    if stopwords is None:
        stopwords = []

    sent1 = [w.lower() for w in sent1]
    sent2 = [w.lower() for w in sent2]

    all_words = list(set(sent1 + sent2))

    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)

    # build the vector for the first sentence
    for w in sent1:
        if w in stopwords:
            continue
        vector1[all_words.index(w)] += 1

    # build the vector for the second sentence
    for w in sent2:
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1

    return 1 - cosine_distance(vector1, vector2)
```

Funkcja *sentence_similarity* jest wykorzystywana w algorytmie generowania streszczenia tekstu do obliczania podobieństwa między dwoma zdaniami. Funkcja przyjmuje dwa argumenty *sent1* i *sent2*, reprezentujące dwa porównywane zdania, oraz opcjonalny argument *stopwords*, który zawiera listę słów, które mają być pominięte w czasie obliczeń.

Funkcja działa na zasadzie obliczenia cosinusowej odległości między wektorami reprezentującymi słowa w dwóch zdaniach. Najpierw obliczane są wektory słów dla obu zdań, które zawierają informacje o liczbie wystąpień poszczególnych słów w każdym zdaniu. Pomijane są w tym miejscu słowa znajdujące się na liście *stopwords* pobranej z biblioteki *nltk*. Następnie obliczana jest odległość cosinusowa między tymi wektorami, a wynik ten jest odejmowany od 1, aby uzyskać wartość podobieństwa między dwoma zdaniami.

Funkcja *sentence_similarity* zwraca wartość podobieństwa między dwoma zdaniami jako liczbę z zakresu od 0 do 1. Im liczba jest bliżej wartości 1 tym zdania są bardziej podobne do siebie.

Funkcja build_similarity_matrix służy do budowania macierzy podobieństwa między zdaniami w tekście:

```
def build_similarity_matrix(sentences, stop_words):
    # Create an empty similarity matrix
    similarity_matrix = np.zeros((len(sentences), len(sentences)))

    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2:  # ignore if both are same sentences
                continue
            similarity_matrix[idx1][idx2] = sentence_similarity(sentences[idx1],
sentences[idx2], stop_words)

    return similarity_matrix
```

Pierwszym krokiem funkcji jest stworzenie pustej macierzy podobieństwa. Każdej kombinacji pary zdań w tekście przypisywana jest wartość podobieństwa między nimi (wykorzystanie funkcji sentence_similarity), a wartości te są zapisywane w macierzy podobieństwa. Pomijane jest obliczanie podobieństwa między tymi samymi zdaniami (*idx1 = idx2*). Na końcu funkcja zwraca pełną macierz podobieństwa między zdaniami, która będzie wykorzystana w dalszych krokach algorytmu podsumowywania tekstu.

Web scraper

Do napisania web scrapera wykorzystana została biblioteka Jsoup, która służy do parsowania i przetwarzania dokumentów HTML.

Działanie stworzonego web scrapera można opisać w następujących krokach:

1) Ustalenie adresu bazowego, z którego pobierane są linki do artykułów
2) Odwiedzenie strony z artykułami i pobranie linków do znajdujących się na niej artykułów
3) Odwiedzenie poszczególnych linków i pobranie interesujących nas informacji (tytuł artykułu, główny obraz oraz treść samego artykułu)

Artykuły pobierane są z kilku portali informacyjnych, a ich treść napisana jest w języku angielskim.

Fragmenty kodu odpowiedzialne za wyszukiwanie i pobieranie linków ze strony:

```
Document document = Jsoup.connect(baseUrl + "/news").get();

Elements aElements = document.getElementsByTag("a");

Set<String> links = new HashSet<>();
for (Element e : aElements) {
    String link = e.attr("href");

    if (!link.contains("/news/")) continue;

    if (!link.contains(baseUrl)) {
        link = baseUrl + link;
    }

    links.add(link);
}

return links;
```

Powyższy kod wyszukuje na podanej stronie wszystkie linki znajdujące się w dokumencie. Wszystkie adresy, które nie zawierają „/news/” są pomijane. Jeżeli w linkach brakuje adresu bazowego to zostaje on dodany i ostatecznie otrzymujemy listę zawierającą adresy do wszystkich artykułów znajdujących się na stronie, która była przeszukiwana.

Następnie dla każdego elementu w liście wykonywana jest kolejna funkcja, która jest odpowiedzialna za pobranie tytułu, treści oraz zdjęcia z podanej strony. Fragment tej funkcji prezentuje się następująco:

```
Document document = Jsoup.connect(link).get();

Element possibleHeading = document.getElementById("main-heading");
if(possibleHeading == null)
    return Optional.empty();
```

```
String heading = possibleHeading.text();

Elements textBlocks = document.getElementsByAttributeValue("data-component", "text-
block");

StringBuilder sb = new StringBuilder();
for (Element textBlock : textBlocks) {
    sb.append(textBlock.getElementsByTag("p").get(0).text()).append(" ");
}

if(sb.isEmpty())
    return Optional.empty();
sb.deleteCharAt(sb.length() - 1); // remove space in end of content

Element pictureElement = document.getElementsByTag("picture").first();

if (pictureElement != null) {
    Element img = pictureElement.getElementsByTag("img").first();
    if (img != null) {
        String src = img.attr("src");
        if(!src.isBlank()) {
           return Optional.of(new News(heading, sb.toString(), link, src));
        }
    }
}
return Optional.of(new News(heading, sb.toString(), link));
```

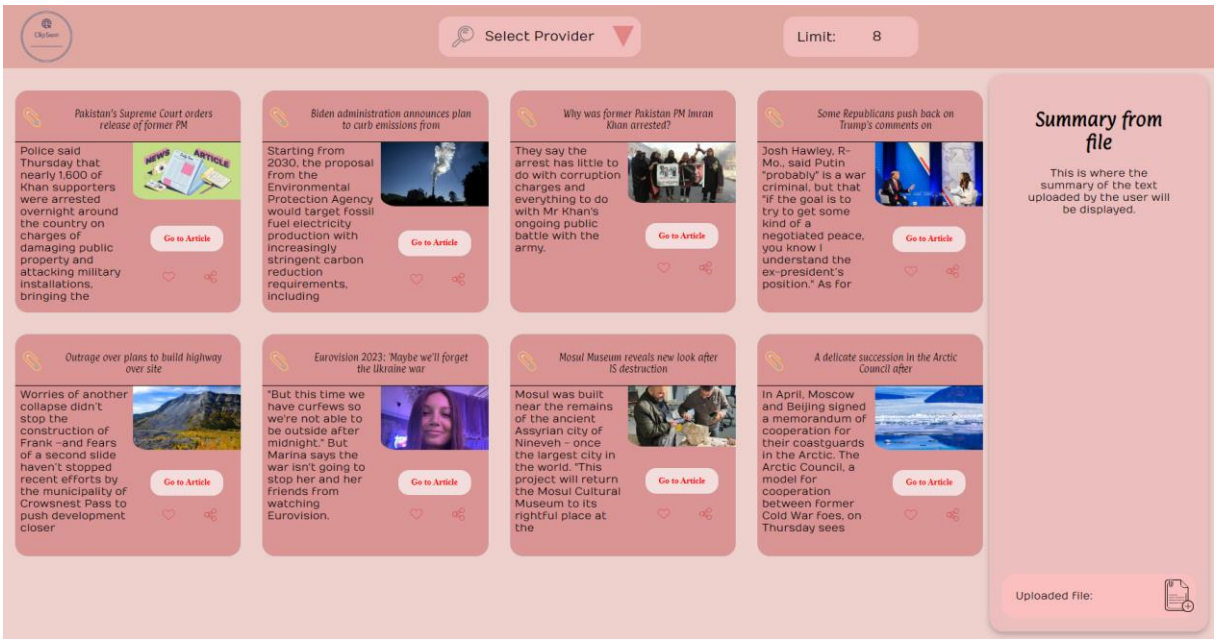
Po dostaniu się pod podany adres, najpierw wyciągnięty zostaje tytuł artykułu. W tym przypadku informacja ta dostępna jest w elemencie o id = „main-heading”. Następnie poszukiwane i łączone w jeden tekst są wszystkie paragrafy, które są częścią artykułu. Na końcu pozostało tylko zapisanie adresu do głównego zdjęcia dla danego artykułu. Jeżeli okaże się, że na danej stronie nie ma któregoś z poszukiwanych elementów, to strona ta jest pomijana, gdyż prawdopodobnie nie jest to strona zawierająca artykuł.

Po odwiedzeniu wszystkich linków otrzymywana jest lista zawierająca informacje na temat pobranych artykułów. Fragment takiej listy prezentuje się następująco (dla „content” prezentowany jest tylko początek artykułu):


```
[
  {
    "title": "Imran Khan: Pakistan's Supreme Court rules arrest was illegal",
    "content": "Pakistan's Supreme Court has ruled that former prime minister Imran
Khan's …"
    "link": "https://www.bbc.com/news/world-asia-65561807",
    "urlToImage":
"https://ichef.bbci.co.uk/news/976/cpsprodpb/571B/production/_129699222_bbcikcourt.
jpg"
  },
  {
    "title": "US border crisis: El Paso readies for rise in crossings as end of
Title 42 looms",
    "content": "A record number of migrants were recently apprehended at the US-
Mexico border in a single day…",
    "link": "https://www.bbc.com/news/world-us-canada-65552877",
    "urlToImage":
"https://ichef.bbci.co.uk/news/976/cpsprodpb/16B97/production/_129697039_us_migrant
_expulsions-nc.png"
  },
    ...
```


Interfejs użytkownika i przykład działania

Po wejściu na stronę, użytkownikowi ukazuje się strona główna aplikacji:



Użytkownik otrzymuje dwie opcje do wyboru: wybranie strony, z której artykuły zostają pobierane (lista rozwijana na górze ekranu) lub załadowanie ręcznie dokumentu do streszczenia (prawy dolny róg). Jeżeli użytkownik zdecyduje się na automatyczne załadowanie i streszczenie artykułów, to finalnie otrzyma streszczone artykuły z wybranego portalu informacyjnego.

Domyślnie, po wejściu na stronę, użytkownikowi prezentowane są losowe artykuły z danego dnia, z różnych źródeł. Aby pobrać artykuły z jednej ze stron informacyjnych, należy wybrać odpowiednie źródło z listy rozwijanej. Dodatkowo jest możliwość określenia liczby wyświetlanych artykułów na stronie (pole „Limit").

Jeżeli użytkownik posiada artykuł w formacie txt, wystarczy, że naciśnie przycisk *Upload file* i wybierze odpowiedni plik. Streszczony tekst umieszczony jest w polu znajdującym się po prawej stronie.

Przykład:
Oryginalny artykuł: https://www.bbc.com/news/world-65527160

Otrzymane streszczenie:

The present is the past," he said. But I don't know whether King Charles is going to do an apology without the British state. Mr Gonsalves has said he would like to try again. Mr Gonsalves added he would welcome an apology from both King Charles and the British government on the legacy of slavery. And I welcome that.

Podsumowanie

Wnioski

Aplikacja pozwala na łatwe i szybkie przetwarzanie długich tekstów na ich krótkie podsumowania, co pozwala użytkownikom na zaoszczędzenie czasu i poznanie kluczowych informacji, bez potrzeby czytania całego tekstu.

Realizacja projektu obejmowała wiele etapów, począwszy od zdefiniowania wymagań i funkcjonalności, poprzez implementację algorytmów do skracania tekstu, aż do stworzenia aplikacji internetowej. Narzędzie pozwala na pobieranie artykułów z różnych serwisów informacyjnych i generowanie ich skróconych wersji, co daje użytkownikom dostęp do najważniejszych informacji w jednym miejscu.

Możliwe rozszerzenia

Projekt dotyczący narzędzia do tworzenia podsumowań tekstu ma wiele możliwych rozszerzeń, które mogą poprawić jego funkcjonalność i zwiększyć zadowolenie użytkowników. Poniżej przedstawiono kilka możliwych rozszerzeń:

- dodanie funkcji tłumaczenia na inne języki pozwoliłoby na dostęp do informacji dla użytkowników, którzy nie znają języka oryginalnego tekstu.
- opracowanie bardziej zaawansowanych algorytmów do skracania tekstu, pozwoliłoby na uzyskanie lepszych podsumowań i bardziej precyzyjnego przedstawienia informacji.
- dodanie funkcji personalizacji na podstawie preferencji użytkowników, pozwoliłoby na dostarczanie rekomendacji artykułów bardziej dopasowanych do ich potrzeb.
- dodanie funkcjonalności udostępniania podsumowań artykułów na platformach społecznościowych pozwoliłoby na szybsze i łatwiejsze udostępnianie informacji dla większej liczby osób.

Usprawnienia w algorytmie

- Dodanie zabezpieczenia dla przypadku zbyt krótkiego tekstu. Jeśli tekst był za krótki, algorytm próbował dodawać do wyniku więcej zdań niż było dostępne.
- Dodanie opcji wystarczającej długości tekstu. Jeśli podsumowanie go osiągnie więcej zdań, nie jest dodawane.

Brakujące elementy

Rzeczy, które były zaplanowane, ale nie zostały wykonane:

- Interfejs dla urządzeń mobilnych
- Polubienia artykułów
- Udostępnianie artykułu
- Osobne okno dla każdego artykułu z możliwością zobaczenia pełnego streszczenia
- Wsparcie dla innych formatów pliku niż txt, np. doc, docx

Bibliografia

- Dubey, Praveen, 2020, Understand Text Summarization and create your own summarizer in python, Towards Data Science, [dostęp: 08.05.2023], https://towardsdatascience.com/understand-text-summarization-and-create-your-own-summarizer-in-python-b26a9f09fc70 https://github.com/edubey/text-summarizer
- Accern, NLP Text Summarization: Benefits & Use Cases, https://accern.com/blog/nlp-text-summarization/ [dostęp: 08.05.2023]