Politechnika Krakowska
Wydział Informatyki i Telekomunikacji

Narzędzie do tworzenia podsumowań tekstu

Rafał Faliński
Andrii Banyk
Paweł Dąbrowa
Krystyn Brzyski

Wstęp:

Podsumowanie można zdefiniować jako zadanie stworzenia zwięzłego i płynnego podsumowania przy jednoczesnym zachowaniu kluczowych informacji i ogólnego znaczenia.

Na przykład, podczas podsumowywania blogów po wpisie na blogu pojawiają się dyskusje lub komentarze, które są dobrym źródłem informacji pozwalającym określić, które części bloga są krytyczne i interesujące.

W podsumowaniu artykułu naukowego znajduje się znaczna ilość informacji, takich jak cytowane artykuły i informacje z konferencji, które można wykorzystać do zidentyfikowania ważnych zdań w oryginalnym artykule.

W tym zadaniu chcielibyśmy stworzyć narzędzie, które będzie właśnie tworzyło podsumowanie tekstu.

Rozwinięcie:

Analiza napisanego programu:

Funkcja sentence_similarity służy do obliczania podobieństwa dwóch zdań. Oba zdania są konwertowane na małe litery i wszystkie słowa są wyodrębniane do listy. Następnie tworzone są wektory dla obu zdań. Na koniec zwracana jest różnica między odległością kosinusową obu wektorów

```
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)

    for w in sent1:
        if w in stopwords:
            continue
        vector1[all_words.index(w)] += 1

    for w in sent2:
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1

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

Następnie użyliśmy funkcje read_article (file_name), która służy do otwierania plików o nazwie podanej jako argument. Następnie dzieli tekst w pliku na zdania, które są zapisywane w liście. Każde zdanie jest sprawdzane za pomocą wyrażeń regularnych, aby usunąć wszystkie znaki niebędące literami, a następnie podzielone na słowa. Na końcu lista jest zwracana.

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

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

    return sentences
```

Następnie użyta została funkcja build_similarity_matrix (sentences, stop_words), która tworzy pustą macierz podobieństwa. Następnie oceniamy podobieństwa między danymi zdaniami. Jeśli zdania są identyczne, pomijana jest ta iteracja. W przeciwnym razie współczynnik podobieństwa jest obliczany za pomocą funkcji sentence_similarity i zapisywany w macierzy podobieństwa. Na końcu macierz podobieństwa jest zwracana.

```
def build_similarity_matrix(sentences, stop_words):
    similarity_matrix = np.zeros((len(sentences), len(sentences)))

    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2:
                continue
            similarity_matrix[idx1][idx2] =
sentence_similarity(sentences[idx1], sentences[idx2], stop_words)

    return similarity_matrix
```

Aby wygenerować podsumowanie tekstu wystarczy użyć funkcji generate_summary(). Funkcja pobiera dwa argumenty: nazwę pliku ze źródłowym tekstem oraz liczbę topowych zdań do wygenerowania podsumowania (domyślnie 5). Najpierw funkcja pobiera listę słów z biblioteki NLTK, które zostaną użyte do obliczenia macierzy podobieństwa zdań. Następnie funkcja wczytuje tekst i dzieli go na zdania. Kolejno macierz podobieństwa zdań jest obliczana z użyciem listy słów. Po tym, jak macierz jest utworzona, funkcja używa algorytmu PageRank do oceny wagi każdego zdania w tekście. Na samym końcu, funkcja wybiera topową liczbę zdań i tworzy podsumowanie tekstu.

```
def generate_summary(file_name, top_n=5):
   nltk.download("stopwords")
   stop_words = stopwords.words('english')
   summarize_text = []
   sentences = read_article(file_name)
   sentence_similarity_martix =
build_similarity_matrix(sentences, stop_words)
   sentence_similarity_graph =
nx.from_numpy_array(sentence_similarity_martix)
   scores = nx.pagerank(sentence_similarity_graph)
   ranked_sentence = sorted(((scores[i], s) for i, s in
enumerate(sentences)), reverse=True)
   print("Indexes of top ranked_sentence order are ",
ranked_sentence)

   for i in range(top_n):
       summarize_text.append(" ".join(ranked_sentence[i][1]))
   print("Summarize Text: \n", ". ".join(summarize_text))
```

Podsumowanie:

Projekt udało się wykonać, program generuje krótkie podsumowania tekstu, co było głównym zamysłem wykonywanego zadania.

Tutaj krótkie podsumowanie tekstu wystawione przez program:

Summarize Text:
Envisioned as a three-year collaborative program, Intelligent Cloud Hub will support around 100 institutions with AI infrastructure, course content and curriculum, developer support, development tools and give students access to cloud and AI services. The company will provide AI development tools and Azure AI services such as Microsoft Cognitive Services, Bot Services and Azure Machine Learning.According to Manish Prakash, Country General Manager-PS, Health and Education, Microsoft India, said, "With AI being the defining technology of our time, it is transforming lives and industry and the jobs of tomorrow will require a different skillset

Bibliografia:

- https://github.com/edubey/text-summarizer