AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ GEOLOGII, GEOFIZYKI I OCHRONY ŚRODOWISKA

KATEDRA GEOINFORMATYKI I INFORMATYKI STOSOWANEJ

Projekt dyplomowy

Aplikacja do zarządzania zbiorami danych

Dataset management application

Autor: Monika Hertel

Kierunek studiów: Inżynieria i Analiza Danych

Opiekun pracy: dr Paweł Oleksik

Kraków, 2024

Spis treści

Wstęp

W czasach przeciążonych informacją,

Cel i zakres projektu

Projekt ma na celu przedstawienie możliwości systemu ułatwiającego użytkownikowi proces zbierania i segregowania informacji. Poniższa praca skupi się na ogólnym koncepcie aplikacji oraz szczegółowym opisie implementacji modułu tworzącego streszczenia.

1 Zagadnienia teoretyczne

1.1 Problem tworzenia streszczeń

(JA TUTAJ ROBIĘ NIENADZOROWANY TEXTRANK JAK COŚ)
Streszczenie w każdej pracy naukowej jest jej ważną częścią. Osoby poszukujące informacji na dany temat nie są w stanie przetworzyć ilości dostępnych informacji. Ma ono na celu przekazać kluczowe informacje o czytanym dokumencie, aby czytelnik mógł ocenić czy dany artykuł zawiera informacje mu przydatne.

W kontekście uczenia maszynowego kondensowanie treści, w celu stworzenia kontekstu, opiera się o obliczanie poziomu istotności dla każdego zdania. [1]. Automatyzacja tego procesu dąży do ułatwienia autorom tworzenia prac, poprzez skrócenie czasu, którego wymaga kreacja abstraktu. Systemy ATS (ang. *Automatic Text Summarization*) są jednym z cięższych wyzwań sztucznej inteligencji, dotyczących przetwarzania języka naturalnego [2]. Podejścia do tworzenia tych systemów, możemy podzielić na ekstraktywane, abstrakcyjne i hybrydowe.

1.1.1 Metody ekstraktywne a abstrakcyjne

Większość badań nad systemami ATS skupia się na użyciu metod ekstraktywnych, starając się przy tym uzyskać zwięzłe i kompletne streszczenia. Podejście ekstraktywne polega na wybraniu najważniejszych zdań z całego dokumentu, a długość uzyskanego wyniku zależy od wartości stopnia kompresji [3].

Streszczenia stworzone z użyciem metod abstraktywnych zawierają zdania, które nie znajdują się w orginalnym tekście. System musi w pewnym stopniu "rozumieć" tekst, aby móc poprawnie zinterpretować dokument. Abstraktywność wymaga implementacji bardziej złożonych algorytmów przetwarzania języka naturalnego.

Istnieje również podział zadania podsumowania na nienadzorowane i nadzorowanie. Nienadzorowane tworzą streszczenia tylko na podstawie danych wejściowych, czyli jedynie zawartości wprowadanego dokumentu. Do optymalnego wyboru zdań, takie systemy implementują metody natury heurystycznej. Z tego powodu są one odpowiednie do przetwarzania danych na bierząco.

Sposób nadzorowany wymaga przeprowadzenia fazy trenowania modelu, która wymaga wprowadzenia opisanego zbioru treningowego. Takie zbiory powinny po-

siadać docelowe postacie streszczeń otrzymane z pełnego tekstu dokumentu. Taki proces jest trudny do przeprowadzenia na większej ilości tekstów.

1.1.2 Ocena poprawności abstraktu

Zazwyczaj ingerencja człowieka jest wymagana przy ewaluacji wytworzonego streszczenia. Treść jest sprawdzana pod kątem poprawności gramatycznej, składni czy całościowej spójności. Taka ewaluacja wymaga dużego nakładu pracy, a przy większych projektach jest praktycznie niemożliwe. Dlatego możliwość zautomatyzowania tego procesu jest wręcz wymagana.

Jednymi z pierwszych metod ewaluacji automatycznej były metryki takie jak podobieństwo cosinusowe (*ang. cosine similarity*), *unit overlap* czy miara najdłuższego wspólnego podzdania (*ang. longest common subsequence*). Te elementy zostały później skondensowane w zbiór kryteriów opisanych akronimem *ROUGE* (*ang. Recall-Oriented Understudy of Gisting Evaluation*) [4]. Poniższy segment przedstawia poszczególne komponenty zestawu *ROUGE*, to jest kryteria *ROUGE-N*, *ROUGE-L* oraz *ROUGE-S*.

ROUGE-N mierzy poziom pokrycia n-gramów, czyli ciągłych sekwencji n słów, pomiędzy wytworzonym tekstem a tekstem źródłowym. Najczęstszym przypadkiem użycia tego kryterium jest ewaluacja poprawności gramatycznej. Miary *ROUGE-1* oraz *ROUGE-2* należą do miar *ROUGE-N* i w kolejności kożystają z unigramów i bigramów. Tymczasem *ROUGE-L* jest wyznaczana na podstawie porównania najdłuższych wspólnych sekwencji słów w zdaniach, występujących w streszczeniu wygenerowanym i wzorcowym. *ROUGE-S* działa na tej samej zasadzie co *ROUGE-2* lecz uwzględnia w swoim działaniu struktury skip-bigram, który jest rozszerzeniem definicji bigramów o możliwość zawierania w sobie maksymalnie jednego przedimka.

1.2 Tworzenie aplikacji webowych

W proces tworzenia aplikacji wchodzi wiele elementów. Wybór odpowiednich narzędzi jest jednym nich.

1.3 Ekstrakcja tekstu plików PDF

Pliki PDF są powszechnie przyjętym standardem przechowywania informacji. Format PDF jest oparty o strukturę binarnego formatu plików, zoptymalizowanego pod kątem wysokiej wydajności odczytu wizualnego. Zawierają w sobie informacje o strukturze dokumentu, takie jak zawartość tekstowa, grafiki czy użyta czcionka. Są one zoptymalizowane pod drukowanie. Niezaszyfrowane PDF mogą być w całości reprezentowane z użyciem wartości bitowych, odpowiadających części zbioru znaków zdefiniowanego w *ANSI X3.4-1986*, symbole kontrolne oraz puste znaki. Wizualnie jednak nie są one ograniczone do zbioru znaków ASCII.

Format PDF separuje informacje dotyczące samego znaku a jego wizualną reprezentacją. Jest to rozróżnienie na znak pisarski i glif, gdzie grafem jest jednostką tekstu a glif, jednostką graficzną. Glif informuje o położeniu znaków na stronie dokumentu, jego czcionce i innych elementach wyglądu.

Otrzymanie zawartości pliku może być wykonane za pomocą wydobycia elementów PDF z strumienia pliku lub z użyciem analizy obrazów, na przykład technologi optycznego rozpoznawania znaków OCR (*ang. Optical Character Recogintion*). Podstawowym zadaniem systemu OCR jest konwersja dokumentów w dane możliwe do edytowania czy wyszukiwania. Techniki oparte o analizę obrazów są bezpośrednio zależne od jakości wprowadzonego elementu. OCR będzie skuteczny w sytuacji gdzie plik zawiera w sobie jedynie tekst i jest obrazem binarnym [5].

Ekstrahowanie danych z strumienia pliku, wiąże się z kilkoma problemami. Biorąc pod uwagę możliwość że plik pdf może posiadać różne kodowanie takie jak *UTF-8*, *ASCII* czy *Unicode*, możemy doświadczyć utraty informacji spowodowanej schematem kodowania pliku. Automatyczna ekstrakcja zawartości polega na selekcji znaków znajdujących się pomiędzy zdefiniowanymi słowami kluczowymi. Pliki PDF są przystosowane do drukowania, z tego powodu reprezentacja tekstu w strumieniu może się znacząco różnić od tej na stronie. Ponieważ pozycje znaków na poszczególnych stronach są absolutne, przedstawienie w strumieniu bierze pod uwagę spacje między elementami.

Niezależne od sposobu pobierania informacji, nie jest możliwe zagwarantowanie ich poprawności względem dokumentu wejściowego. Brak formalnej definicji struktury, przynajmniej jeżeli chodzi o artykuły naukowe, uniemożliwia stworzenie

uniwersalnego algorytmu ekstrakcji.

2 Implementacja

2.1 Projekt systemu

Aplikacja została zaprojektowana z myślą o użytkownikach będących w posiadaniu dużej ilości plików z danymi. Głównym celem jest automatyzacja procesu segregowania i przetwarzania informacji. System etykietuje otrzymane pliki poprzez ekstrakcje słów kluczowych (pliki zawierające tekst) i daje możliwość wyszukiwania plików w bardziej optymalny sposób. Otrzymane dokumenty są kategoryzowane ze względu na słowa klucze. Po otrzymaniu elementów o podobnej strukturze i zawartości, są one łączone ze sobą. Informacja o takim połączeniu jest umieszczana w bazie danych, tak aby przy odczycie jednego z elementów, drugi był sugerowany jako następny plik do wyświetlenia.

2.2 Struktura aplikacji

Aplikacja została zbudowana z myślą o zapewnieniu interfejsu użytkownika przy użyciu JavaScript React, serwera HTTP opartego na Flask Python, oraz bazy danych MongoDB, która jest bazą typu NoSQL. Zintegrowana całość jest uruchamiana z wykorzystaniem pakietu Electron, co umożliwia stworzenie aplikacji natywnej, zachowując przy tym funkcje przeglądarki oraz pozwalając na dostęp do zasobów systemowych.

2.2.1 Javascript React

Jest to biblioteka pozwalająca na budowanie interaktywnych interfejsów użytkownika. Główną koncepcją React są komponenty, czyli samodzielne, hermetyczne jednostku interfejsu. Pozwala to na reakcje na zmiany wywołane przez użytkownika lub sam system i przy tym automatyczne aktualizowanie tych stron. React używa składni JSX (*Javascript XML*), który jest rozszerzeniem składni Javascript. Integruje ona kod Javascript z deklaratywnym opisem struktury interfejsu.

2.2.2 Python Flask

Flask to lekki i elastyczny framework, pozwalający na szybkie tworzenie aplikacji internetowych przy minimalnej ilości kodu. Dostarcza on jedynie niezbędne narzędzia,

stawiając tym na dużą swobodę implementacji innych funkcji.

PRZEPISAĆ [Najważniejszą jego cechą, z punktu widzenia tej pracy jest fakt, że jest on biblioteką języka python. W pierwszej fazie testów używanych algorytmów, były one tworzone z pomocą tego języka. Python jest językiem często używanym w szerokim spektrum tematu, jakim jest sztuczna inteligencja. Tutaj serwer jest traktowany jako narzędzie umożliwiające kożystanie z bibliotek języka python.]

Funkcja serwera jest uruchamiana w sytuacji gdy na podany adres URL zostanie wysłany komunikat. Informacja o aktywującym adresie jest przekazywana z użyciem dekoratora *.route()*. Rysunek X przedstawia przykład takiego wywołania.

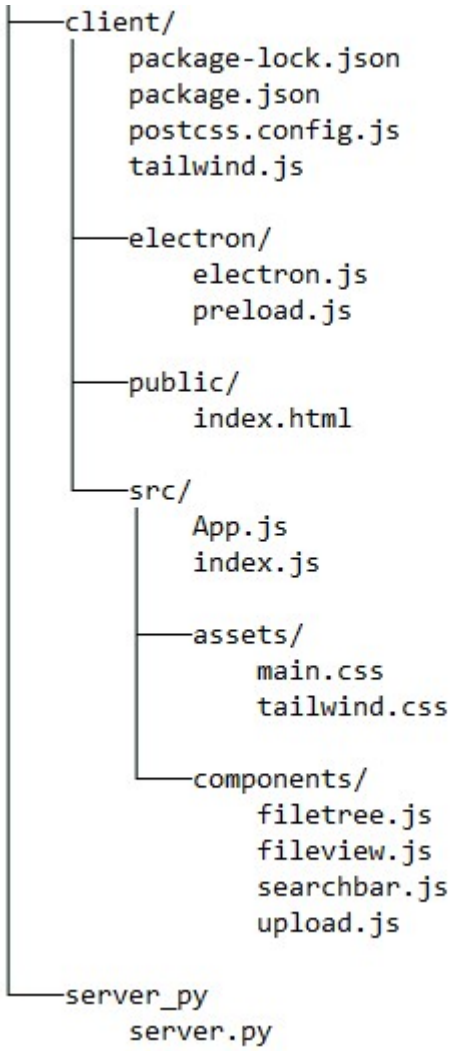
```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Rysunek 1: Przykładowa implementacja funkcji serwera

2.2.3 Electron

Biblioteka Electron jest projektem open-source umożliwiającym tworzenie aplikacji natywnych używając technologii web. Standard tworzenia aplikacji z pomocą tej biblioteki jest oparty o komunikację międzyprocesową (*ang. IPC, Inter-process communication*), która jest wynikiem implementacji izolacji kontekstu. Izolacja kontekstu odpowiada za separację procesu renderowania strony od głównego procesu aplikacji w środowisku Node.js. Zwiększa to bezpieczeństwo aplikacji, negując możliwość dostępu do użytych API z strony.

System posiada jeden główny proces *"main"*, którego celem jest tworzenie aplikacji i kierowanie podprocesami, oraz wiele procesów odpowiedzialnych za renderowanie zawartości strony zwanych procesami *"renderer"*. Biblioteka umożliwia również dodanie skryptów, które działają przed renderowaniem strony (ang. *preload scripts*). Otrzymują one dodatkowe możliwości w porównaniu plików *renderer*, poprzez posiadanie uprawnień do korzystania z Node.js APIs. Mogą one bezpiecznie udostępnić części API dla procesów strony.



Rysunek 2: Uogólniona struktura projektu

2.3 Architektura systemu

Baza danych

2.4 Schematy poszczególnych funkcjonalności

(dodać screeny z postmana jak wywołuje jakąś funkcje) Poniżej zaprezentowane są funkcjonalności końcowej aplikacji na przykładzie plików o rozszerzeniu *.pdf*. (można zrobić opisy funkcjonalności, nawet tych nie zaimplementowanych razem ze schematami, a potem cały opis modułu streszczeń)

2.4.1 Dodawanie pliku

Zachowanie systemu przy dodaniu plików tekstowych.

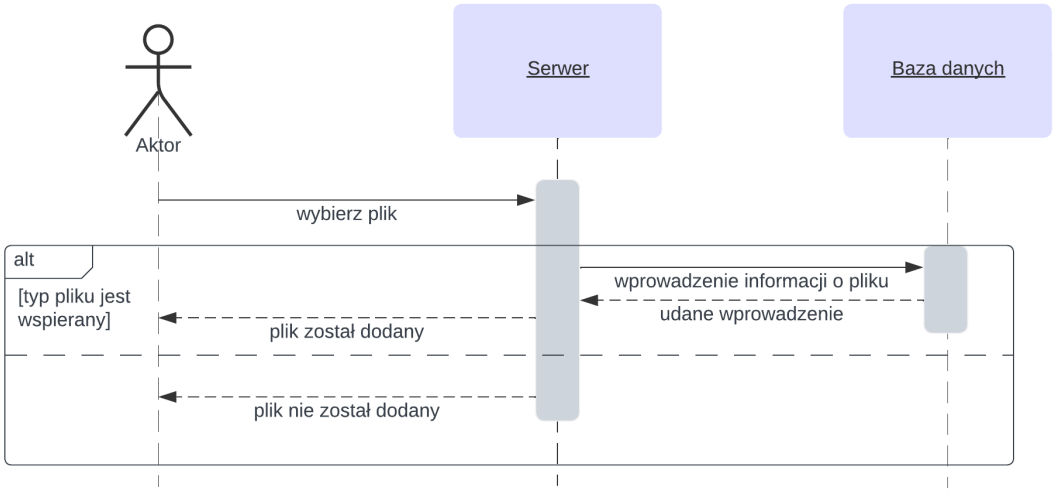
1. Gdy użytkownik wybierze plik, aplikacja przesyła ścieżkę pliku do serwera,
2. System ekstraktuje z pliku całą zawartość z pomocą biblioteki języka Python *PdfMiner*,
3. Z zawartości zostają wyciągnięte słowa klucze. Tworzenie tzw. tagów odbywa się z pomocą biblioteki *yake* oraz metody *KeywordExtractor()*. Przyjmuje ona następujące parametry:
    - *lan* - język danego tekstu,
    - *n* - maksymalna ilość słów w jednym tagu,
    - *deduplim* - szansa na powtóżenie się słów w różnych słowach kluczach,
    - *top* - ilość elementów wyjściowych

```
lang = "en"
max_ngram_size = 3
deduplication_treshold = 0.5
num_keywords = 6
kw_extract = yake.KeywordExtractor(lan=lang,n=max_ngram_size,
                                   dedupLim=deduplication_treshold,
                                   top=num_keywords, features=None)

keywords = kw_extract.extract_keywords(text)
```

Rysunek 3: przykład użycia *KeywordExtractor()*

4. w tym samym czasie z pomocą biblioteki *sumy* oraz funkcji *TextRankSummarizer*, jest tworzone streszczenie z użyciem algorytmu *TextRank*,
5. jeżeli plik pdf posiada zakładki (ang. *outlines*) to metoda *.get_outlines()* ekstraktuje je i z wyniku możemy otrzymać tytuł dokumentu i nadać plikowi taką nazwę. a jeżeli ich nie ma to tytułem pliku zostaje bazowa nazwa pliku,
6. ostatecznie wszystkie informacje są zbierane i przesyłane do bazy danych.

Model TextRank zastosowany w projekcie, jest algorytmem rankingowym opartym na grafach [6]. Operuje on na zasadach “głosowania” i “rekomendacji”. Jest on używany do tworzenia streszczeń metodą ekstraktywną i nienadzorowaną. Dla naszego systemu, nie jest możliwe wytworzenie odpowiedniego zbioru treningowego, aby móc zastosować metody nadzorowane.



Rysunek 4: Schemat sekwencyjny dodawania pliku

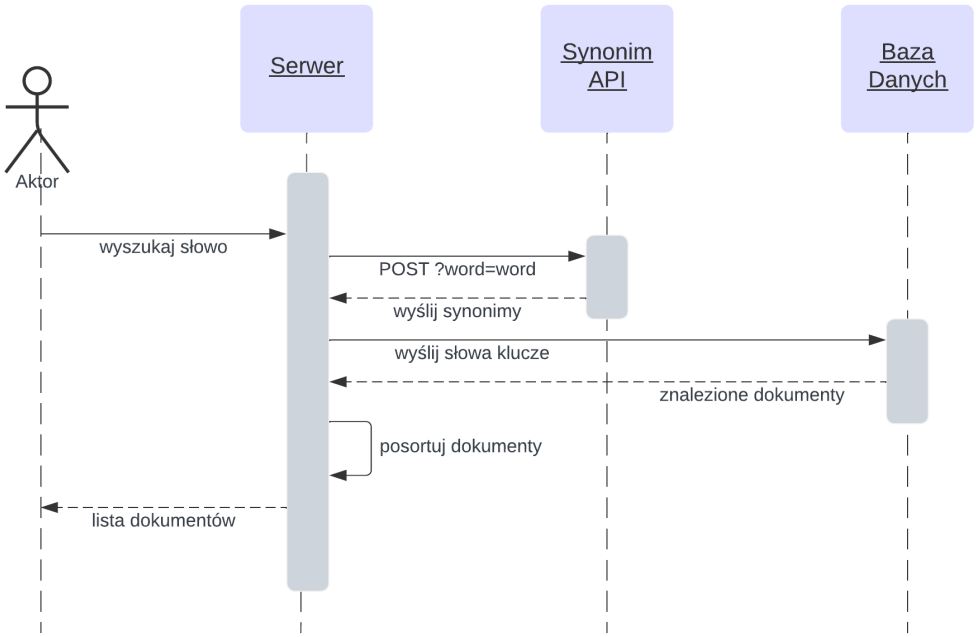
2.4.2 Wyszukiwanie i wyświetlanie plików

Użytkownik ma możliwość wysukiwania plików po słowach kluczach lub tytule pliku. Tutaj przydatne jest użycie API generującego synonimy dla wyszukiwanego słowa. System zachowuje się w poniżej opisany sposób.

1. aplikacja przesyła komunikat z wyszukiwanym słowem,
2. serwer używając API synonimów pobiera 5 najbliższych słów do słowa szukanego przypisując im ranking,
3. serwer przesyła osobne komunikaty do tabeli *file_properties* w naszej bazie danych, zawierające osobno słowo klucz oraz synonimy
4. baza zwraca informacje dotyczące znalezionych dokumentów oraz słowo klucz,
5. serwer zwraca listę plików użytkownikowi, posortowane zgodnie z rankingiem

2.4.3 Asocjacja plików o podobnej tematyce

Poniższa sekcja przedstawia proces łączenia dokumentów na podstawie wygenerowanych słów kluczowych.



Rysunek 5: Schemat sekwencyjny wyszukiwania pliku

2.5 Testy funkcjonalne

Jeżeli chodzi o testowanie tworzenia streszczeń to nie jest możliwa automatyzacja tego. Nie jesteśmy w stanie przewidzieć jakie zdania będą wchodziły w skład wygenerowanego abstraktu. Czynnik ludzki w tym przypadku jest wymagany do oceny poprawności danego algorytmu. (wait to jak oni sprawdzają to przy metodach abstrakcyjnych????)

jak będzie czas to mogę przetestować ocr i tersseract

2.6 TEMP Idealna aplikacja

DOBRA JEŻELI ROBIMY STRESZCZENIA TO ŻEBY NAJPIERW BYŁ NIENADZOROWANY TYP ROBIENIA STRESZCZEŃ A POTEM JAK PODZIELIMY PLIKI NA KATEGORIE TO WTEDY PRÓBOWAĆ ROBIĆ Z DANYMI WEJŚCIOWYMI, CZYLI NASZE WCZEŚNIEJ DODANE PLIKI (THE BLIND LEADING THE BLIND)

jeszcze można wygenerować nasze streszczenia jako osobny plik; wykrywanie języka dokumentu by się przydało

3 Możliwości rozwoju i wykorzystania aplikacji

Podsumowanie i wnioski

W sumie to nie wiem co jest celem tej pracy ani co mam podsumować,

Spis rysunków

Literatura

[1] B. Mutlu, E. A. Sezer, and M. A. Akcayol, "Candidate sentence selection for extractive text summarization," *Information Processing & Management*, vol. 57, no. 6, p. 102359, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457320308542

[2] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed, "Automatic text summarization: A comprehensive survey," *Expert Systems with Applications*, vol. 165, p. 113679, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420305030

[3] M. Gambhir and V. Gupta, "Recent automatic text summarization techniques: a survey," *Artificial Intelligence Review*, vol. 47, no. 1, pp. 1–66, Jan 2017. [Online]. Available: https://doi.org/10.1007/s10462-016-9475-9

[4] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013

[5] R. Mithe, S. Indalkar, and N. Divekar, "Optical character recognition," *International journal of recent technology and engineering (IJRTE)*, vol. 2, no. 1, pp. 72–75, 2013.

[6] R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, D. Lin and D. Wu, Eds. Barcelona, Spain: Association

for Computational Linguistics, Jul. 2004, pp. 404–411. [Online]. Available: https://aclanthology.org/W04-3252