AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Search Engine

Dokumentacja

Radosław Rolka
Metody obliczeniowe w nauce i technologii
Informatyka WI AGH, II rok

2024

Spis treści

1 Informacje wstępne

Celem było utworzenie silnika wyszukiwarki oraz utworzenie pochodnej wersji działającej z SVD i aproksymacją low rank oraz interfejsu użytkownika.

2 Konfiguracja wstępna

Jako dane początkowe wybrałem:

Parametr	Wartość
START_CRAWL	Chimichanga
DOMAIN	https://en.wikipedia.org/wiki/
MAX_PAGES	100000
BAG_OF_WORDS	50000

Tabela 1: Parametry Wyszukiwarki

3 Pozyskiwanie danych

Do pobrania artykułów z wikipedii w liczbie 100.000 wykorzystałem BeautifulSoup4

```python
def get_data(config):
    counter = 0
    queue = deque([config["START_CRAWL"]])
    visited = set()

    while queue and counter < config["MAX_PAGES"]:
        page = queue.popleft()
        if page in visited:
            continue
        visited.add(page)
        counter += 1
        print(f"Fetching {page} ...{counter}/{config['MAX_PAGES']}")
        response = requests.get(urljoin(config["DOMAIN"], page))
        soup = BeautifulSoup(response.text, "html.parser")

        with open(f"./data/articles/{quote(page, safe='')}.txt", "w") as f:
            for tag in soup.find_all(["h1", "h2", "h3", "h4", "h5", "p", "a", "li"]):
                f.write(tag.get_text() + "\n")

        for anchor in soup.select("#bodyContent a"):
            href = anchor.get("href")
            if href and href.startswith("/wiki/") and ":" not in href:
                queue.append(href[6:])
```

Kod źródłowy 1: Sprawdzenie czy drzewo zawiera punkt.

4 Przetwarzanie artykułów

Aby ujednolicić słowa występujące w artykułach wykonałem:

- sprowadzenie do postaci małych liter
- usunięcie słów przystankowych

- usunięcie końcówek fleksyjnych
- usunięcie znaków interpunkcyjnych
- usunięcie zbyt krótkich słów (>3)

```
text = f.read()
            text = text.lower()
            text = re.sub(r'[^a-z]', ' ', text)
            text = re.sub(r'\s+', ' ', text)
            text = text.strip()
            words = text.split(' ')
            words = [word for word in words if word not in stopwords.words('english')]
            words = [stemmer.stem(word) for word in words]
            words = [word for word in words if len(word) >= 3]
```

Kod źródłowy 2: Sprawdzenie czy drzewo zawiera punkt.

5 Słownik słów kluczowych

Następnie policzyłem słowa i utworzyłem słownik słów kluczowych o rozmiarze 50.000 słów

```
local_dict = {word: words.count(word) for word in set(words)}
            for word, v in local_dict.items():
                if word not in all_file_dict:
                    all_file_dict[word] = v
                else:
                    all_file_dict[word] += v

    sorted_dict = sorted(all_file_dict.items(), key=lambda x: x[1], reverse=True)
    sorted_dict = sorted_dict[:config['BAG_OF_WORDS']]
    sorted_dict = dict(sorted_dict)
```

Kod źródłowy 3: Sprawdzenie czy drzewo zawiera punkt.

6 Tworzenie macierzy rzadkiej

Przy tworzeniu macierzy uprzednio pomnożyłem elementy o współczynnik IDF, a następnie znormalizowałem wektory artykułów.

```
def initialize_matrix(config):
    def idf(word):
        return np.log(config['MAX_PAGES'] / word_in_diff_file[word])

    list_of_articles = []
    list_of_words = list(main_dict.keys())
    matrix = sparse.csc_matrix((0,0), dtype=np.float64)
    for file in os.listdir('./data/dicts'):
        list_of_articles.append(file[:-5])
        with open(f'./data/dicts/{file}', 'r') as f:
            local_dict = json.load(f)
            local_dict = {word: local_dict[word] * idf(word) for word in local_dict if word in ma
            column = [local_dict[word] if word in local_dict else 0 for word in list_of_words]
            column = column / np.linalg.norm(column)
            matrix = sparse.hstack([matrix, sparse.csc_matrix(column).transpose()]).tocsc()
```

Kod źródłowy 4: Sprawdzenie czy drzewo zawiera punkt.

7 Macierz SVD

```
def initialize_svd(matrix, k):
    U, s, V = svds(matrix, k=k)
```

Kod źródłowy 5: Sprawdzenie czy drzewo zawiera punkt.

8 Wprowadzanie zapytania

Przed wyszukiwaniem przetwarzam zapytanie w sposób jednakowy co przetwarzałem zawartość artykułów.

```
def process_query(text):
    text = text.lower()
    text = re.sub(r'[^a-z]', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    text = text.strip()
    words = text.split(' ')
    words = [word for word in words if word not in stopwords.words('english')]
    words = [PorterStemmer().stem(word) for word in words]
    words = [word for word in words if len(word) >= 3]
    return words
```

Kod źródłowy 6: Sprawdzenie czy drzewo zawiera punkt.

A następnie wykonywałem działania jak na wektorach artykułów w macierzy.

```
def to_vector(words, list_of_words):
    query_vector = np.zeros(len(list_of_words))
    for word in words:
        if word in list_of_words:
            query_vector[list_of_words.index(word)] += 1
    query_vector = query_vector / np.linalg.norm(query_vector)
    return query_vector
```

Kod źródłowy 7: Sprawdzenie czy drzewo zawiera punkt.

9 Wyszukiwanie z cosinusową miarą podobieństwa

Aby otrzymać wyniki mnożę wektor zapytania z macierza rzadką

```
def search(query_vector, matrix, k):
    query_vector = query_vector / np.linalg.norm(query_vector)
    result = query_vector @ matrix
    articles_idx = np.argpartition(result, result.size - k)[-k:]
    return zip(articles_idx[np.argsort(result[articles_idx])][::-1], result[articles_idx[np.argso
```

Kod źródłowy 8: Sprawdzenie czy drzewo zawiera punkt.

10 Wyszukiwanie low rank

Aby otrzymać wyniki mnorzę wektor zapytania z kolejnymi macierzami wynikowym SVD.

```
def search(query_vector, U, s, V, results_num):
    query_vector = query_vector / np.linalg.norm(query_vector)
    qA = ((query_vector@U)@np.diag(s))@V
    result = sorted(enumerate(qA), key=lambda x: x[1], reverse=True)
    return result[:results_num]
```

Kod źródłowy 9: Sprawdzenie czy drzewo zawiera punkt.

11 Serwer Flask

Tworząc serwer Flask ładuję wsyzstkie dane do configu oraz zezwalam na zapytania CORS.

```
app = Flask(__name__)
CORS(app)
with open('config.json', encoding='utf-8') as f:
    config = json.load(f)

app.config.update(config)
app.config['MATRIX'] = sparse.load_npz('./data/matrix.npz')
for k in [50, 100, 200]:
    with open(f'./data/svd_{k}/U.npy', 'rb') as f:
        U = np.load(f)
    with open(f'./data/svd_{k}/s.npy', 'rb') as f:
        s = np.load(f)
    with open(f'./data/svd_{k}/V.npy', 'rb') as f:
        V = np.load(f)
    app.config[f'SVD_{k}'] = [U, s, V]
with open('./data/list_of_articles.txt', 'r', encoding='utf-8') as f:
    app.config['LIST_OF_ARTICLES'] = f.read().split('\n')
with open('./data/list_of_words.txt', 'r', encoding='utf-8') as f:
    app.config['LIST_OF_WORDS'] = f.read().split('\n')
```

Kod źródłowy 10: Sprawdzenie czy drzewo zawiera punkt.

12 API Serwera

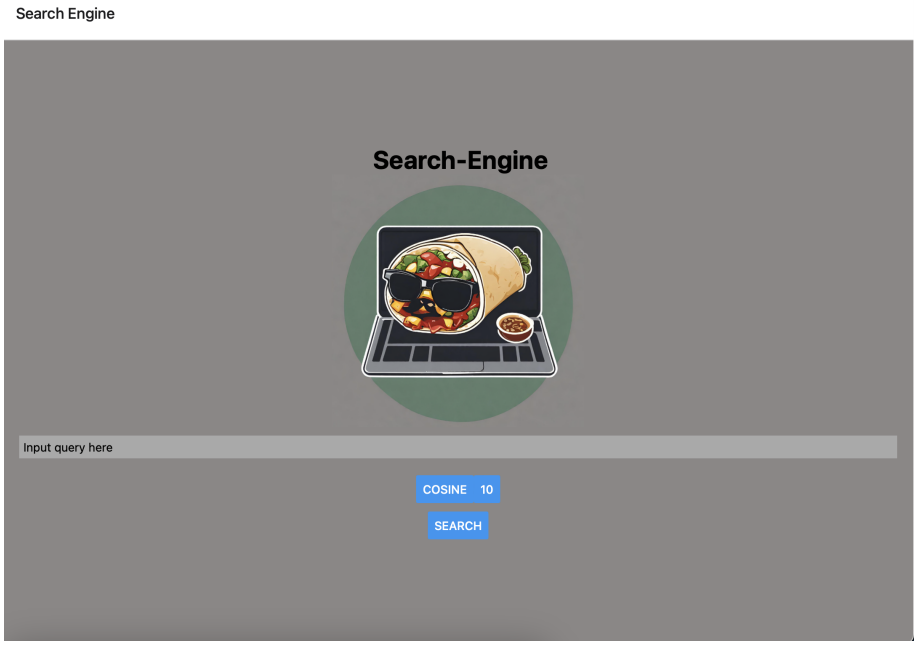
Serwer wykorzystuje funkcje opisane powyżej w sposób następujący:

```
def cosine():
    query_param = request.args.get('query', default = '', type = str)
    results_num = request.args.get('results_num', default = 10, type = int)
    processed_query = query_processing.process_query(query_param)
    vector = query_processing.to_vector(processed_query, app.config['LIST_OF_WORDS'])
    result = cosine_search.search(vector, app.config['MATRIX'], results_num)
    response = [[app.config['LIST_OF_ARTICLES'][name], round(100*(0 if np.isnan(perc) else perc),
    return jsonify({'result': response}), 200
```

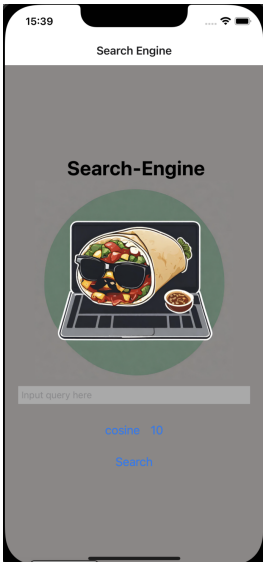
Kod źródłowy 11: Sprawdzenie czy drzewo zawiera punkt.

13 React Native front-end

Dzięki wykorzystaniu React-Native napisana aplikacja działa na wszystkich platformach z wykorzystaniem tego samego kodu.



Rysunek 1: Aplikacja webowa

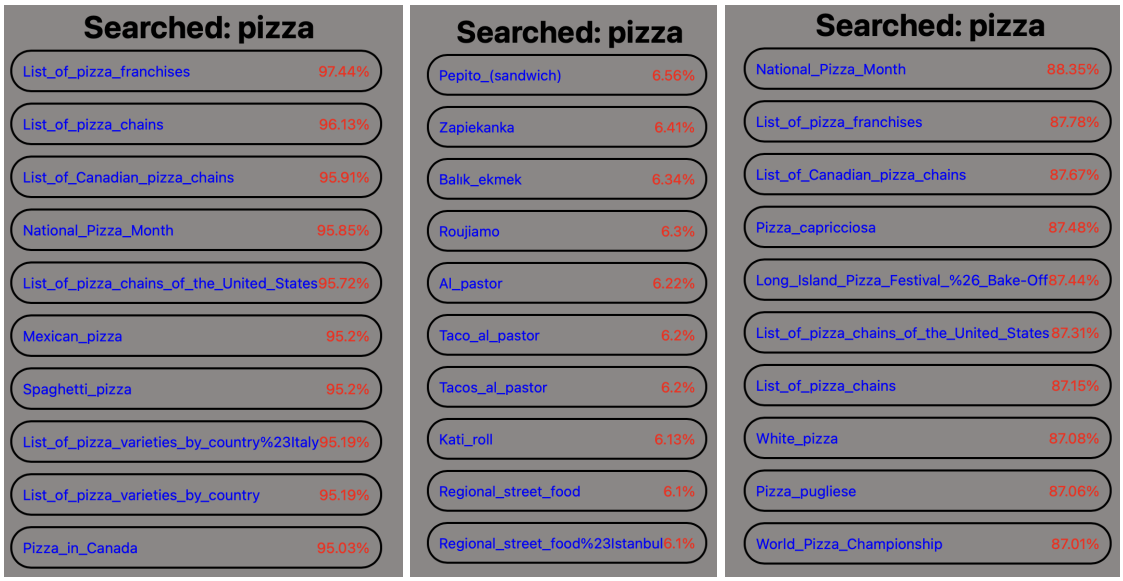


Rysunek 2: Aplikacja IOS

14 Porównanie działania z szumem i bez

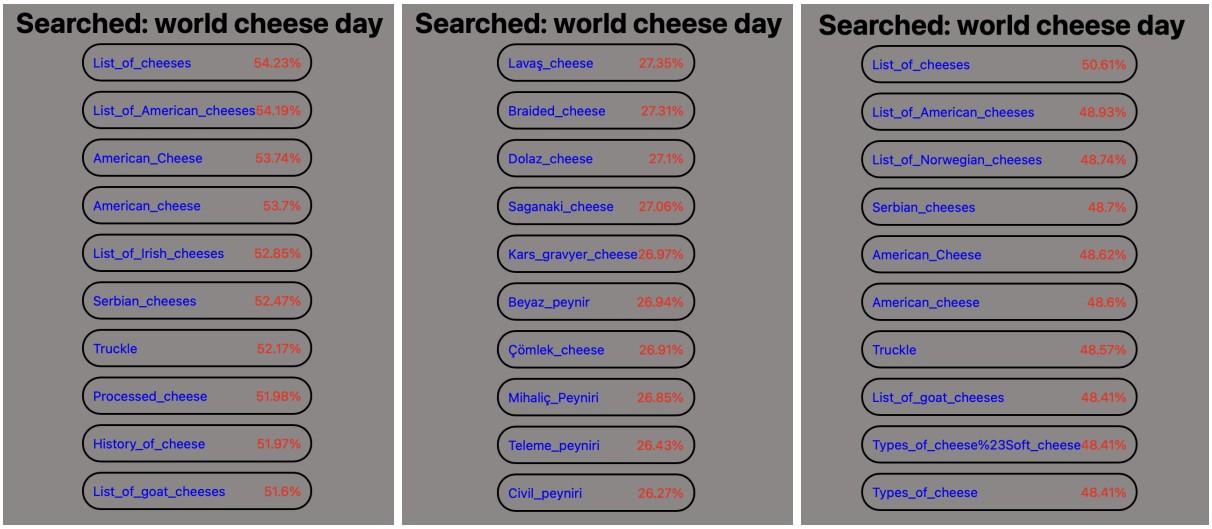
Testy będą dla wersji bez szumu oraz zaszumionych z parametrem k=50 oraz k=200:

- "Pizza"



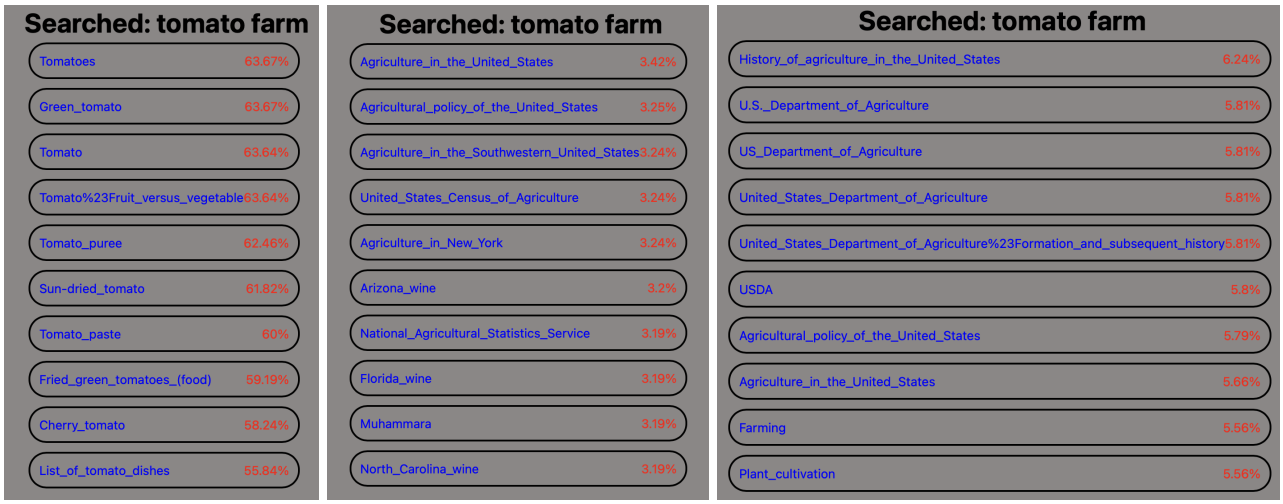
Rysunek 3: (a) Bez szumu (b) k=50 (c) k=200

- "World cheese day"



Rysunek 4: (a) Bez szumu (b) k=50 (c) k=200

- "Tomato Farm"



Rysunek 5: (a) Bez szumu (b) k=50 (c) k=200

15 Podsumowanie

Podsumowując, wyniki eksperymentów wykazały, że wszystkie warianty radzą sobie zadaniami. Wersja bez szumu wykazuje surowość w ocenie słów kluczowych, co może być istotne w określonych kontekstach. Natomiast wersja zaszumiona 200 okazała się lepsza od wersji zaszumionej 50, szczególnie przy dużej ilości słów, sugerując, że większe dodanie szumu może poprawić wydajność w pewnych scenariuszach.