



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	3
2	Konfiguracja wstępna	3
3	Pozyskiwanie danych	3
4	Przetwarzanie artykułów	3
5	Słownik słów kluczowych	4
6	Tworzenie macierzy rzadkiej	4
7	Macierz SVD	5
8	Wprowadzanie zapytania	5
9	Wyszukiwanie z cosinusową miarą podobieństwa	5
10	Wyszukiwanie low rank	5
11	Serwer Flask	6
12	API Serwera	6
13	React Native front-end	6
14	Porównanie działania z szumem i bez	8
15	Podsumowanie	9

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

```

1 def get_data(config):
2     counter = 0
3     queue = deque([config["START_CRAWL"]])
4     visited = set()
5
6     while queue and counter < config["MAX_PAGES"]:
7         page = queue.popleft()
8         if page in visited:
9             continue
10        visited.add(page)
11        counter += 1
12        print(f"Fetching {page}...{counter}/{config['MAX_PAGES']}")
13        response = requests.get(urljoin(config["DOMAIN"], page))
14        soup = BeautifulSoup(response.text, "html.parser")
15
16        with open(f"./data/articles/{quote(page, safe='')} .txt", "w") as f:
17            for tag in soup.find_all(["h1", "h2", "h3", "h4", "h5", "p", "a", "li"]):
18                f.write(tag.get_text() + "\n")
19
20        for anchor in soup.select("#bodyContent a"):
21            href = anchor.get("href")
22            if href and href.startswith("/wiki/") and ":" not in href:
23                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)

```

1 text = f.read()
2     text = text.lower()
3     text = re.sub(r'^a-z', ' ', text)
4     text = re.sub(r'\s+', ' ', text)
5     text = text.strip()
6     words = text.split(' ')
7     words = [word for word in words if word not in stopwords.words('english')]
8     words = [stemmer.stem(word) for word in words]
9     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

```

1 local_dict = {word: words.count(word) for word in set(words)}
2     for word, v in local_dict.items():
3         if word not in all_file_dict:
4             all_file_dict[word] = v
5         else:
6             all_file_dict[word] += v
7
8     sorted_dict = sorted(all_file_dict.items(), key=lambda x: x[1], reverse=True)
9     sorted_dict = sorted_dict[:config['BAG_OF_WORDS']]
10    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.

```

1 def initialize_matrix(config):
2     def idf(word):
3         return np.log(config['MAX_PAGES'] / word_in_diff_file[word])
4
5     list_of_articles = []
6     list_of_words = list(main_dict.keys())
7     matrix = sparse.csc_matrix((0,0), dtype=np.float64)
8     for file in os.listdir('./data/dicts'):
9         list_of_articles.append(file[:-5])
10        with open(f'./data/dicts/{file}', 'r') as f:
11            local_dict = json.load(f)
12            local_dict = {word: local_dict[word] * idf(word) for word in local_dict if word in ma
13            column = [local_dict[word] if word in local_dict else 0 for word in list_of_words]
14            column = column / np.linalg.norm(column)
15            matrix = sparse.hstack([matrix, sparse.csc_matrix(column).transpose()]).tocsc()

```

Kod źródłowy 4: Sprawdzenie czy drzewo zawiera punkt.

7 Macierz SVD

```
1 def initialize_svd(matrix, k):
2     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.

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

Kod źródłowy 6: Sprawdzenie czy drzewo zawiera punkt.

A następnie wykonywałem działania jak na wektorach artykułów w macierzy.

```
1 def to_vector(words, list_of_words):
2     query_vector = np.zeros(len(list_of_words))
3     for word in words:
4         if word in list_of_words:
5             query_vector[list_of_words.index(word)] += 1
6     query_vector = query_vector / np.linalg.norm(query_vector)
7     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

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

Kod źródłowy 8: Sprawdzenie czy drzewo zawiera punkt.

10 Wyszukiwanie low rank

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

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

Kod źródłowy 9: Sprawdzenie czy drzewo zawiera punkt.

11 Serwer Flask

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

```

1 app = Flask(__name__)
2 CORS(app)
3 with open('config.json', encoding='utf-8') as f:
4     config = json.load(f)
5
6 app.config.update(config)
7 app.config['MATRIX'] = sparse.load_npz('./data/matrix.npz')
8 for k in [50, 100, 200]:
9     with open(f'./data/svd_{k}/U.npy', 'rb') as f:
10         U = np.load(f)
11         with open(f'./data/svd_{k}/s.npy', 'rb') as f:
12             s = np.load(f)
13             with open(f'./data/svd_{k}/V.npy', 'rb') as f:
14                 V = np.load(f)
15         app.config[f'SVD_{k}'] = [U, s, V]
16 with open('./data/list_of_articles.txt', 'r', encoding='utf-8') as f:
17     app.config['LIST_OF_ARTICLES'] = f.read().split('\n')
18 with open('./data/list_of_words.txt', 'r', encoding='utf-8') as f:
19     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:

```

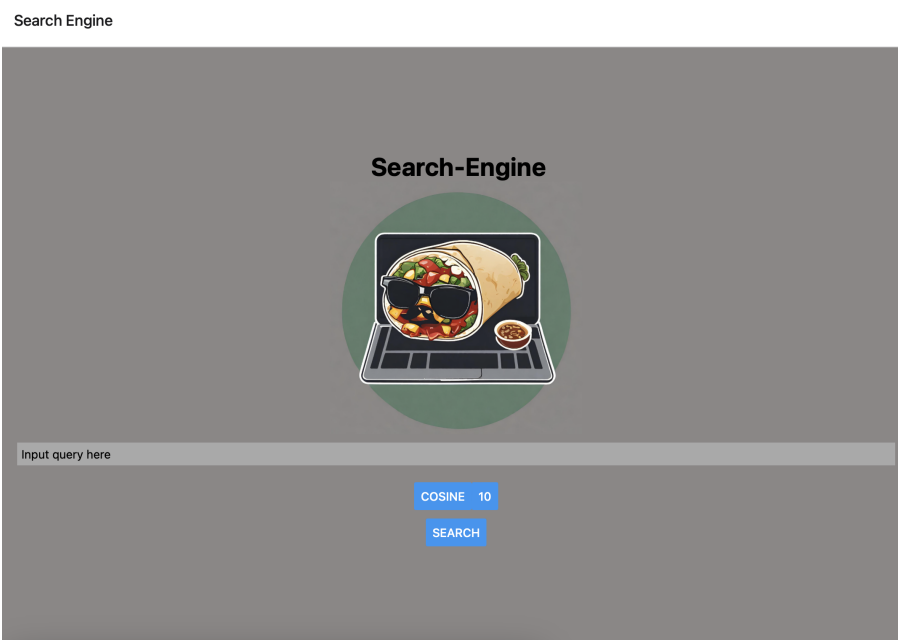
1 def cosine():
2     query_param = request.args.get('query', default = '', type = str)
3     results_num = request.args.get('results_num', default = 10, type = int)
4     processed_query = query_processing.process_query(query_param)
5     vector = query_processing.to_vector(processed_query, app.config['LIST_OF_WORDS'])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

```

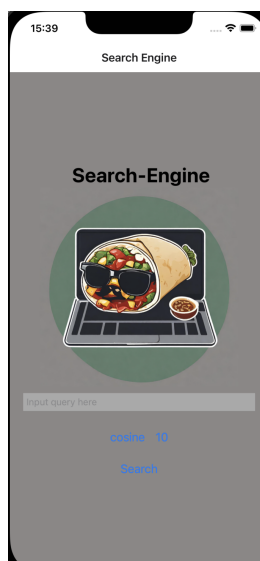
Kod źródłowy 11: Sprawdzenie czy drzewo zawiera punkt.

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"

Searched: pizza	Searched: pizza	Searched: pizza
List_of_pizza_franchises 97.44%	Pepito_(sandwich) 6.56%	National_Pizza_Month 88.35%
List_of_pizza_chains 96.13%	Zapiekanka 6.41%	List_of_pizza_franchises 87.78%
List_of_Canadian_pizza_chains 95.91%	Balik_ekmek 6.34%	List_of_Canadian_pizza_chains 87.67%
National_Pizza_Month 95.85%	Roujiamo 6.3%	Pizza_capricciosa 87.48%
List_of_pizza_chains_of_the_United_States 95.72%	Al_pastor 6.22%	Long_Island_Pizza_Festival_%26_Bake-Off 87.44%
Mexican_pizza 95.2%	Taco_al_pastor 6.2%	List_of_pizza_chains_of_the_United_States 87.31%
Spaghetti_pizza 95.2%	Tacos_al_pastor 6.2%	List_of_pizza_chains 87.15%
List_of_pizza_varieties_by_country%23Italy 95.19%	Kati_roll 6.13%	White_pizza 87.08%
List_of_pizza_varieties_by_country 95.19%	Regional_street_food 6.1%	Pizza_pugliese 87.06%
Pizza_in_Canada 95.03%	Regional_street_food%23Istanbul 6.1%	World_Pizza_Championship 87.01%

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

- "World cheese day"

Searched: world cheese day	Searched: world cheese day	Searched: world cheese day
List_of_cheeses 54.23%	Lava_cheese 27.35%	List_of_cheeses 50.61%
List_of_American_cheeses 54.19%	Braided_cheese 27.31%	List_of_American_cheeses 48.93%
American_Cheese 53.74%	Dolaz_cheese 27.1%	List_of_Norwegian_cheeses 48.74%
American_cheese 53.7%	Saganaki_cheese 27.06%	Serbian_cheeses 48.7%
List_of_Irish_cheeses 52.85%	Kars_gravyer_cheese 26.97%	American_Cheese 48.62%
Serbian_cheeses 52.47%	Beyaz_peynir 26.94%	American_cheese 48.6%
Truckle 52.17%	Çömlek_cheese 26.91%	Truckle 48.57%
Processed_cheese 51.98%	Mihaliç_Peyniri 26.85%	List_of_goat_cheeses 48.41%
History_of_cheese 51.97%	Teleme_peyniri 26.43%	Types_of_cheese%23Soft_cheese 48.41%
List_of_goat_cheeses 51.6%	Civil_peyniri 26.27%	Types_of_cheese 48.41%

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

- "Tomato Farm"

Searched: tomato farm	Searched: tomato farm	Searched: tomato farm
Tomatoes 63.67%	Agriculture_in_the_United_States 3.42%	History_of_agriculture_in_the_United_States 6.24%
Green_tomato 63.67%	Agricultural_policy_of_the_United_States 3.25%	U.S._Department_of_Agriculture 5.81%
Tomato 63.64%	Agriculture_in_the_Southwestern_United_States 3.24%	US_Department_of_Agriculture 5.81%
Tomato%23Fruit_versus_vegetable 63.64%	United_States_Census_of_Agriculture 3.24%	United_States_Department_of_Agriculture 5.81%
Tomato_puree 62.46%	Agriculture_in_New_York 3.24%	United_States_Department_of_Agriculture%23Formation_and_subsequent_history 5.81%
Sun-dried_tomato 61.82%	Arizona_wine 3.2%	USDA 5.8%
Tomato_paste 60%	National_Agricultural_Statistics_Service 3.19%	Agricultural_policy_of_the_United_States 5.79%
Fried_green_tomatoes_(food) 59.19%	Florida_wine 3.19%	Agriculture_in_the_United_States 5.66%
Cherry_tomato 58.24%	Muhammara 3.19%	Farming 5.56%
List_of_tomato_dishes 55.84%	North_Carolina_wine 3.19%	Plant_cultivation 5.56%

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.