Search Engine

Wojciech Kaźmierczak

1. Wstęp

Celem laboratorium było napisanie prostej wyszukiwarki internetowej działającej na kilkudziesięciu tysiącach artykułów oraz kilkudziesięciu tysiącach słów w słowniku projekt można podzielić na kilka głównych etapów:

a) Napisanie webcrawler'a
b) Przetworzenie teksu z wyszukanych stron
c) Utworzenie słownika wszystkich słów oraz macierzy rzadkiej wystąpień słów w artykułach
d) Testowanie low rank approximation (SVD)
e) Napisanie prostego frontend'u

2. Web crawler

Jako artykuły, które będzie można wyszukać w mojej wyszukiwarce wybrałem artykuły z angielskiej Wikipedii. W tym celu użyłem biblioteki **wikipedia** z python'a. Liczba artykułów pobranych to dokładnie **11 877.** Kod za to odpowiedzialny:

```
def fetch_article_links(start_page, max_articles):
    article_links = set()
    article_links.add(wikipedia.page(start_page).url)
    visited_links = set()

    while len(article_links) < max_articles:
        current_link = article_links.pop()
        if current_link in visited_links:
            continue
        visited_links.add(current_link)

        try:
            response = requests.get(current_link)
            soup = BeautifulSoup(response.content, 'html.parser')
            for link in soup.find_all('a', href=True):
                link_href = link['href']
                if link_href.startswith('/wiki/') and ':' not in link_href:
                    article_links.add('https://en.wikipedia.org' + link_href)

        except:
            pass
        time.sleep(0.5)
    return list(article_links)[:max_articles]
```

3. Przetwarzanie tekstu i utworzenie słownika oraz macierzy rzadkiej

W przetwarzaniu tekstu przydatne były biblioteki **bs4** oraz **nltk**, które to czytały zawartość artykułów po czym usuwały **stopwords**. Uznałem, że wystarczające będzie **150** najczęstszych słów z każdego artykułów, aby oddać ich główny (przynajmniej pozorny) sens. Kod za to odpowiedzialny:

```
def fetch_article_content(article_links):
    articles = []
    for link in article_links:
        try:
            print(f"Fetching article: {link}")
            response = requests.get(link)
            soup = BeautifulSoup(response.content, 'html.parser')
            for script in soup(["script", "style"]):
                script.extract()
            text = soup.get_text()
            tokens = word_tokenize(text)
            tokens = [word.lower() for word in tokens]
            stop_words = set(stopwords.words('english'))
            tokens = [word for word in tokens if word.isalnum() and word not in stop_words]
            cnt = Counter(tokens).most_common(150)
            content = [item for item, _ in cnt]
            articles.append((link, content, cnt))
        except:
            pass

    with open("list_of_links_12000_norm.txt", "w") as file:
        for item in articles:
            line = ';'.join(map(str, item))
            file.write(line + '\n')
    return articles


def remove_stopwords(article):
    article = re.sub(r'[^\w\s]', '', article).lower()
    tokens = word_tokenize(article)
    filtered_article = [word for word in tokens if word not in stop_words]
    filtered_article = ' '.join(filtered_article)

    return filtered_article
```

Bez liczenia powtarzających się słów utworzony przeze mnie słownik ma **90 936** wyrazów. Zatem utworzona macierz (przechowywana jako macierz rzadka w pliku o rozszerzeniu „**.pickle**”) ma wymiary **90 936x11 877.** Dodatkowo zastosowana została metoda **inverse document frequency** mająca na celu redukcję znaczenia słów często występujących.

$$IDF\ (w) = \log\left(\frac{N}{n_w}\right)$$

gdzie n_w jest liczbą dokumentów, w których występuje słowo w, a N jest całkowitą liczbą dokumentów.

Utworzenie macierzy rzadkiej na następnej stronie (kod):

```python
def save_articles(articles):
    all_words = dict()
    num_of_files = len(articles)
    nm = Counter()
    bags = []
    idx = 0
    urls = dict()

    for i, (url, content, cnt) in enumerate(articles):
        bags.append(cnt)
        nm.update(Counter(content))
        urls[i] = url
        for word in content:
            if word not in all_words:
                all_words[word] = idx
                idx += 1
        print(f"Article num: {i+1}")

    n = len(all_words)

    word_count = np.array([])
    rows_idx = np.array([])
    cols_idx = np.array([])

    for i in range(num_of_files):
        for elem, count in bags[i]:
            word_count = np.append(word_count, count * np.log(num_of_files / nm[elem]))
            rows_idx = np.append(rows_idx, all_words[elem])
            cols_idx = np.append(cols_idx, i)

    sparse_matrix = csr_matrix((word_count, (rows_idx, cols_idx)), shape=(n, num_of_files))

    # normalize
    sparse_matrix_csr = sparse_matrix.tocsr()
    column_norms = np.sqrt((sparse_matrix_csr.power(2)).sum(axis=0))
    column_norms[column_norms == 0] = 1
    sparse_matrix = sparse_matrix_csr / column_norms

    with open('./pickle_dir/data12000_norm.pickle', 'wb') as file:
        pickle.dump((sparse_matrix, all_words, urls), file)
```

4. Wyszukiwanie

Na początku następuje wczytanie macierzy rzadkiej z pliku po czym następuje obliczenie korelacji pomiędzy wektorem zapytania a kolejnymi kolumnami macierzy rzadkiej. Dzięki znormalizowaniu wektorów cech oraz wektora **q** reprezentującego szukane zapytanie (bag-of-words) możliwe było użycie następującej miary prawdopodobieństwa.

$$|q^T A| = [|\cos\theta_1|, |\cos\theta_2|, \ldots, |\cos\theta_n|]$$

Im większy $\cos\theta_i$ tym bardziej i-ty artykuł pasuje do danego zapytania.
Kod odpowiedzialny za wyszukiwanie (na następnej stronie):

```python
def load_data():
    with open('./pickle_dir/data12000_norm.pickle', 'rb') as file:
        sparse_matrix, all_words, urls = pickle.load(file)
    return sparse_matrix, all_words, urls


def search(query):
    sparse_matrix, all_words, urls = load_data()
    n, num_of_files = sparse_matrix.shape
    q = np.zeros(n)

    for word in query:
        if word in all_words.keys():
            q[all_words[word]] = 1

    if not (q.any() != 0):
        print("None of the words in articles")
        return []

    # normalize
    q /= np.linalg.norm(q)

    # low rank approximation
    # svd = TruncatedSVD(n_components=1000)
    # sparse_matrix = svd.fit_transform(sparse_matrix)

    res = [(elem, i) for i, elem in enumerate(abs(q.T@sparse_matrix))]
    res = [x for x in res if x[0] > 0]
    res = nlargest(100, res, key=lambda x: x[0])
    val = []
    for elem, i in res:
        url = urls[i]
        title = url.split("/")[-1].strip()
        val.append((elem, url, title))
    return val
```

5. Testowanie SVD (Singular Value Decomposition)

Pierwsze próby wykonania SVD podjąłem z wykorzystaniem funkcji **svds**() z biblioteki **scipy** oraz funkcji **svd**() z biblioteki **numpy** jednak za każdym razem brakowało pamięci RAM w moim komputerze (testy dla k =100). Dopiero funkcji **TruncatedSVD** dawała jakiekolwiek wyniki jednak były one gorsze od wyników uzyskanych bez SVD.

Wynik dla zapytania „pear”

Bez SVD

'https://en.wikipedia.org/wiki/Pear',
'https://en.wikipedia.org/wiki/Opuntia',
'https://en.wikipedia.org/wiki/Parapsychology',
'https://en.wikipedia.org/wiki/Survivalism_(life_after_death)',
'https://en.wikipedia.org/wiki/Horseshoe-shaped'

Z SVD k=100

'https://en.wikipedia.org/wiki/Eukaryote',
'https://en.wikipedia.org/wiki/Phytogeography',
'https://en.wikipedia.org/wiki/Glorious_Revolution_in_Scotland',
'https://en.wikipedia.org/wiki/Boomerang_(Central_and_Eastern_European_TV_channel)'
'https://en.wikipedia.org/wiki/CTC_Media', 'https://en.wikipedia.org/wiki/Herbert_Hoover'

Z SVD k=1000

'https://en.wikipedia.org/wiki/Sir_George_Stokes,_1st_Baronet',
'https://en.wikipedia.org/wiki/State_of_Palestine',
'https://en.wikipedia.org/wiki/Pesticide',
'https://en.wikipedia.org/wiki/Human_body#Regional_groups',
'https://en.wikipedia.org/wiki/Isle_Royale'

Intuicyjnie oraz poprzez sprawdzenie liczebności wystąpień słowa pear w zwróconych artykułach najlepsze wyniki daje brak zastosowania SVD, dlatego też tą wersje ostatecznie wybrałem, aby SVD okazało się lepsze musiałbym dysponować większym RAMem.

Drugim rozwiązaniem tego problemu jest wybranie mniejszego zbioru danych, co zrobiłem. Używany zbiór danych do testu wpływu k w SVD na wyniki wyszukiwarki to zbiór o rozmiarze rzędu **25000x1000.**
Pliki odpowiedzialne za przetestowanie SVD **testingSVDs.py** oraz **SVD.py.**

Wyniki dla podzapytania „internet python web”
zaprezentowane na następnej stronie

---------- No SVD ----------

Num of files: 992

Time :0.05024290084838867

Correlations:

[0.38366686567727626, 0.15819511501726707, 0.08637751396735817, 0.06313833048745932, 0.06311027133247915]

['https://en.wikipedia.org/wiki/Wayback_Machine', 'https://en.wikipedia.org/wiki/Animal_Diversity_Web', 'https://en.wikipedia.org/wiki/Food_chain', 'https://en.wikipedia.org/wiki/Sandgrouse', 'https://en.wikipedia.org/wiki/Pterocliformes']

Words in dictionary: 24732

---------- SVD: k=100 ----------

Num of files: 992

Time :1.8081679344177246

Correlations:

[0.059203645929161976, 0.0493505415759277, 0.042863949879926325, 0.03682536845608588, 0.03353134842456235]

['https://en.wikipedia.org/wiki/Four_Noble_Truths', 'https://en.wikipedia.org/wiki/Dukkha', 'https://en.wikipedia.org/wiki/Enlightenment_(Buddhism)', 'https://en.wikipedia.org/wiki/Father_Time', 'https://en.wikipedia.org/wiki/Wayback_Machine']

Words in dictionary: 24732

---------- SVD: k=500 ----------

Num of files: 992

Time :1.48471999168396

Correlations:

[0.38388976071079806, 0.18185129133120084, 0.17641028411354293, 0.13821594636221587, 0.11097941525105537]

['https://en.wikipedia.org/wiki/Wayback_Machine', 'https://en.wikipedia.org/wiki/Main_Page', 'https://en.wikipedia.org/wiki/Johannes_Theodor_Reinhardt', 'https://en.wikipedia.org/wiki/Animal_Diversity_Web', 'https://en.wikipedia.org/wiki/Journal_of_Genetics']

Words in dictionary: 24732

---------- SVD: k=700 ----------

Num of files: 992

Time :1.699296236038208

Correlations:

[0.3839050869521072, 0.1563492397690195, 0.1352817000977719, 0.09449543421429044, 0.08973911175991038]

['https://en.wikipedia.org/wiki/Wayback_Machine', 'https://en.wikipedia.org/wiki/Main_Page', 'https://en.wikipedia.org/wiki/Animal_Diversity_Web', 'https://en.wikipedia.org/wiki/Johannes_Theodor_Reinhardt', 'https://en.wikipedia.org/wiki/Journal_of_Genetics']

Words in dictionary: 24732

---------- SVD: k=850 ----------

Num of files: 992

Time :2.214430809020996

Correlations:

[0.38368106347933706, 0.15720018444465148, 0.0864745193670272, 0.07734314251882018, 0.07174589282611908]

['https://en.wikipedia.org/wiki/Wayback_Machine', 'https://en.wikipedia.org/wiki/Animal_Diversity_Web', 'https://en.wikipedia.org/wiki/Food_chain', 'https://en.wikipedia.org/wiki/Journal_of_Genetics', 'https://en.wikipedia.org/wiki/Dictionary.com']

Words in dictionary: 24732

---------- SVD: k=990 ----------

Num of files: 992

Time :2.2532312870025635

Correlations:

[0.3836668656772765, 0.15819511501726718, 0.08637751396735821, 0.0631383304874594, 0.0631102713324792]

['https://en.wikipedia.org/wiki/Wayback_Machine', 'https://en.wikipedia.org/wiki/Animal_Diversity_Web', 'https://en.wikipedia.org/wiki/Food_chain', 'https://en.wikipedia.org/wiki/Sandgrouse', 'https://en.wikipedia.org/wiki/Pterocliformes']

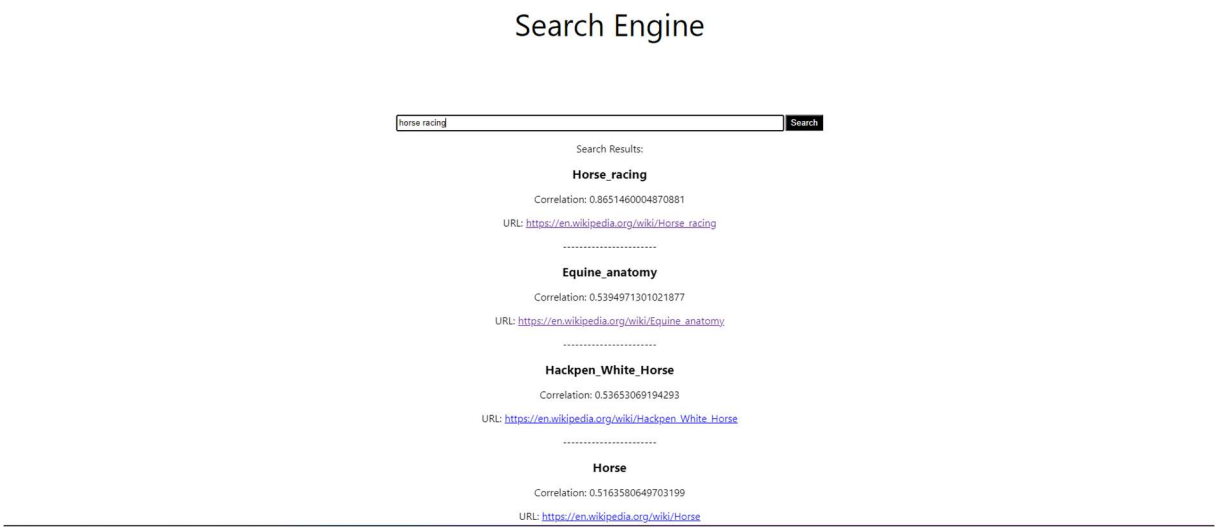
Words in dictionary: 24732

Wedle oczekiwań wraz ze wzrostem k wyniki są coraz bardziej podobne do tych uzyskanych bez SVD można by się spodziewać że najlepsze k pod wyszukiwanie na tym mniejszym zbiorze byłoby k=850, gdyż daje zbliżone wyniki do tych dokładnie uzyskanych, ale też zakłada nieprecyzyjne zapytania użytkownika.
Aby wyszukiwarka działała na owym zbiorze wystarczy w pliku app.py zmienić wczytywany plik (funkcja **load_data()**) z **data12000_norm.pickle** na **data1000_SVD_k850.pickle** lub po k wybraną przez nas liczbę z wyżej uwzględnionych powyżej.
Można wybrać na której macierzy rzadkiej ma działać program wybierając któryś z plików w pickle_dir (wpisując go jako name w load_data() w app.py)

6. Frontend

Aplikacja zrealizowana została przy użyciu React'a (ChakraUI) oraz Flask'a.

Po wprowadzeniu szukanego przez nas zapytania otrzymujemy wyniki z niezerową korelacją posortowane malejąco. Wyświetla się tytuł, korelacja oraz URL. Wyniki otrzymujemy w mniej niż sekundę.



Scrollując w dół możemy zobaczyć resztę wyników.

Dołączam osobno poglądowy film z użytkowania wyszukiwarki.

Uruchomienie aplikacji:

Będąc w folderze backend napisać komendę w terminalu:

```
$ flask run
```

Po czym będąc w folderze frontend napisać komendę w terminalu:

```
$ npm start
```

Aplikacja jest wtedy dostępna pod adresem http://localhost:3000/