Metody Obliczeniowe w Nauce i Technice

Laboratorium 6

Wojciech Łącki

SPIS TREŚCI

Zadanie 1 – Wyszukiwarka 2

ZADANIE 1 – WYSZUKIWARKA

1. Przygotuj duży (> 1000 elementów) zbiór dokumentów tekstowych w języku angielskim (np. wybrany korpus tekstów, podzbiór artykułów Wikipedii, zbiór dokumentów HTML uzyskanych za pomocą *Web crawlera*, zbiór rozdziałów wyciętych z różnych książek).

```
import wikipedia
import json


def save_data(data):
  with open("data.json", "w") as outfile:
    json.dump(data, outfile, indent=4, sort_keys=True)


def get_wikipedia_pages_content(number):
  data = []
  while len(data) != number:
    page = wikipedia.random(1)
    try:
      a = wikipedia.page(page)
      if not any(suspect["title"] == a.title for suspect in data):
        content = str(a.content).replace("\n", " ")
        content = content.replace('"', "'")
        c = content.split("== See Also ==")
        content = c[0]
        c = content.split("== References ==")
        content = c[0]
        data.append({"title": a.title, "content": content})
    except:
      print("ERROR:", a.title)
    print(len(data))
  return data


n = 2000
data = get_wikipedia_pages_content(n)
save_data(data)
```

Powyższe funkcje pobierają i zapisują losowe dokumenty z Wikipedii do pliku.

2. Polecenie: Określ słownik słów kluczowych (termów) potrzebny do wyznaczenia wektorów cech *bag-of-words* (indeksacja). Przykładowo zbiorem takim może być unia wszystkich słów występujących we wszystkich tekstach.

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string
import json

english_stop_words = set(stopwords.words("english"))
punctuation = string.punctuation


def create_dictionary_of_words():
  f = open("data.json", "r")
```

```
  data = json.load(f)
  f.close()
  n = len(data)
  # n = 1001
  dictionary = {}
  for i in range(n):
    text = data[i]["content"].lower()
    words = nltk.word_tokenize(text)
    words = [word for word in words if len(word) > 2]
    words = [word for word in words if ord(word[0]) <= 122 and
    (ord(word[0]) < 48 or ord(word[0]) > 57)]
    words = [word for word in words if word[0] not in punctuation]
    words = [word for word in words if word not in english_stop_words]
    stemmer = PorterStemmer()
    words = [stemmer.stem(word) for word in words]
    for word in words:
      if word in dictionary.keys():
        dictionary[word] += 1
      else:
        dictionary[word] = 1
    print(i)
  return dictionary


def save_dictionary(dictionary):
  with open("dictionary.json", "w") as outfile:
    json.dump(dictionary, outfile, indent=4, sort_keys=True)


dictionary = create_dictionary_of_words()
save_dictionary(dictionary)
```

Dzięki powyższemu kodowi zapiszemy do pliku unię wszystkich słów ze wszystkich dokumentów wraz z ilością wystąpień każdego z nich.

3. Polecenie: Dla każdego dokumentu *j* wyznacz wektor cech *bag-of-words* d_j zawierający częstości występowania poszczególnych słów (termów) w tekście.

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string

nltk.download('stopwords')
nltk.download('punkt')

english_stop_words = set(stopwords.words("english"))
punctuation = string.punctuation


def create_bag_of_words_vector(dictionary, data):
  vector = {}
  for word in dictionary.keys():
    vector[word] = 0
  text = data.lower()
  words = nltk.word_tokenize(text)
  words = [word for word in words if len(word) > 2]
  words = [word for word in words if ord(word[0]) <= 122 and (ord(word[0])
  < 48 or ord(word[0]) > 57)]
  words = [word for word in words if word[0] not in punctuation]
```

```
  words = [word for word in words if word not in english_stop_words]
  stemmer = PorterStemmer()
  words = [stemmer.stem(word) for word in words]
  for word in words:
    vector[word] += 1
  arr = list(vector.values())  # or [vector[key] for key in vector.keys()]
  return arr
```

Powyższa funkcja na podstawie słownika zlicza ilość wystąpień słów w jakimś fragmencie tekstu, czyli u nas będą to pobrane dokumenty z Wikipedii.

4. Polecenie: Zbuduj rzadka macierz wektorów cech *term-by-document matrix* w której wektory cech ułożone są kolumnowo $Am\times n = [d_1|d_2|\ldots|d_n]$ (*m* jest liczba termów w słowniku, a *n* liczba dokumentów).

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string
import json

nltk.download('stopwords')
nltk.download('punkt')

english_stop_words = set(stopwords.words("english"))
punctuation = string.punctuation


def create_bag_of_words_matrix():
  f = open("data.json", "r")
  data = json.load(f)
  f.close()
  f = open("dictionary.json", "r")
  dictionary = json.load(f)
  f.close()
  n = len(data)
  # n = 1001
  matrix = []
  for i in range(n):
    matrix.append({data[i]["title"]: create_bag_of_words_vector(dictionary,
data[i]["content"])})
    print(i)
  return matrix


def create_bag_of_words_vector(dictionary, data):
  vector = {}
  for word in dictionary.keys():
    vector[word] = 0
  text = data.lower()
  words = nltk.word_tokenize(text)
  words = [word for word in words if len(word) > 2]
  words = [word for word in words if ord(word[0]) <= 122 and (ord(word[0])
< 48 or ord(word[0]) > 57)]
  words = [word for word in words if word[0] not in punctuation]
  words = [word for word in words if word not in english_stop_words]
  stemmer = PorterStemmer()
  words = [stemmer.stem(word) for word in words]
  for word in words:
    vector[word] += 1
```

```
  arr = list(vector.values())  # or [vector[key] for key in vector.keys()]
  return arr


def save_dictionary(matrix):
  with open("bag_of_words_vectors.json", "w") as outfile:
    json.dump(matrix, outfile, sort_keys=True)


matrix = create_bag_of_words_matrix()
save_dictionary(matrix)
```

Kod ten zapisuje do pliku macierz (raczej słownik, w kolejnych zadaniach zostanie on przekształcony na macierz), wektorów cech.

5. Polecenie: Przetwórz wstępnie otrzymany zbiór danych mnożąc elementy *bag-of-words* przez *inverse document frequency*. Operacja ta pozwoli na redukcje znaczenia często występujących słów.

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

gdzie n_w jest liczba dokumentów, w których występuje słowo w, a N jest całkowita liczba dokumentów.

```
import json
import math


def count_documents_with_word(data, word_index):
  count = 0
  for document in data:
    amount = list(document.values())[0][word_index]
    if amount > 0:
      count += 1
  return count


def save_dictionary(matrix):
  with open("IDF.json", "w") as outfile:
    json.dump(matrix, outfile, sort_keys=True)


def multiply_by_IDF(data, word_index, idf):
  for document in data:
    for key in document.keys():
      if document[key][word_index] != 0:
        document[key][word_index] *= idf


def calculate_IDF(dictionary, data):
  n = len(data)
  word_index = 0
  for _ in dictionary.keys():
    nw = count_documents_with_word(data, word_index)
    idf = math.log(n / nw)
    multiply_by_IDF(data, word_index, idf)
    word_index += 1
    print(word_index)
```

```
f = open("dictionary.json", "r")
dictionary = json.load(f)
f.close()
f = open("bag_of_words_vectors.json", "r")
data = json.load(f)
f.close()

calculate_IDF(dictionary, data)
save_dictionary(data)
```

Dzięki kodowi umieszczonemu powyżej obliczymy wartość IDF każdego słowa, a następnie mnożymy odpowiednie elementy w macierzy z zadania 4 i znowu zapisujemy do pliku. Zabieg ten pozwala na obniżenie znaczenia słów występujących wiele razy.

6. Polecenie: Napisz program pozwalający na wprowadzenie zapytania (w postaci sekwencji słów) przekształcanego następnie do reprezentacji wektorowej *q* (*bag-of-words*). Program ma zwrócić *k* dokumentów najbardziej zbliżonych do podanego zapytania *q*. Użyj korelacji między wektorami jako miary podobieństwa

$$\cos\theta_j = \frac{q^T d_j}{\|q\|\|d_j\|}$$

```
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string
import json
import math

english_stop_words = set(stopwords.words("english"))
punctuation = string.punctuation


def create_bag_of_words_vector(dictionary, data):
  vector = {}
  for word in dictionary.keys():
    vector[word] = 0
  text = data.lower()
  words = nltk.word_tokenize(text)
  words = [word for word in words if len(word) > 2]
  words = [word for word in words if ord(word[0]) <= 122 and (ord(word[0])
  < 48 or ord(word[0]) > 57)]
  words = [word for word in words if word[0] not in punctuation]
  words = [word for word in words if word not in english_stop_words]
  stemmer = PorterStemmer()
  words = [stemmer.stem(word) for word in words]
  for word in words:
    vector[word] += 1
  arr = list(vector.values())  # or [vector[key] for key in vector.keys()]
  return arr


def get_k_closest_documents(dictionary, idfs, data, k):
  vector = create_bag_of_words_vector(dictionary, data)
  vector_T = np.array(vector).T
  best_k_elements = []
  index = 0
  for document in idfs:
```

```
    for key in document.keys():
      values = document[key]
      cos = (vector_T @ np.array(values)) / (np.linalg.norm(vector) *
      np.linalg.norm(values))
      if len(best_k_elements) < k:
        best_k_elements.append((key, cos))
        best_k_elements = sorted(best_k_elements, key=lambda x: x[1])
      else:
        last = best_k_elements[0][1]
        if cos > last or math.isnan(last):
          best_k_elements[0] = (key, cos)
          best_k_elements = sorted(best_k_elements, key=lambda x: x[1])
    index += 1
    print(index)
  best_k_elements = list(reversed(best_k_elements))
  return best_k_elements


query = input("Search: ")
f = open("dictionary.json", "r")
dictionary = json.load(f)
f.close()
f = open("IDF.json", "r")
idfs = json.load(f)
f.close()
k = 10
documents = get_k_closest_documents(dictionary, idfs, query, k)
print(documents)
```

Powyższa funkcja wyszukuje najbardziej dopasowane dokumenty względem wprowadzonego zapytania.

7. Polecenie: Zastosuj normalizacje wektorów cech d_j i wektora q, tak aby miały one długość 1. Użyj zmodyfikowanej miary podobieństwa otrzymując

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

```
import numpy as np
import string
import json
import time
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.preprocessing import normalize

english_stop_words = set(stopwords.words("english"))
punctuation = string.punctuation


def normalize_matrix(idfs):
  normalized = []
  count = 0
  for document in idfs:
    for key in document.keys():
      normalized_array = normalize([document[key]], norm="l1")[0].tolist()
      for i in range(len(normalized_array)):
        if normalized_array[i] == 0:
          normalized_array[i] = 0
      normalized.append(normalized_array)
    count += 1
```

```
    print(count)
  return normalized


def save_dictionary(matrix):
  with open("IDF_normalized_matrix.json", "w") as outfile:
    json.dump(matrix, outfile, sort_keys=True)


def create_bag_of_words_vector(dictionary, data):
  vector = {}
  for word in dictionary.keys():
    vector[word] = 0
  text = data.lower()
  words = nltk.word_tokenize(text)
  words = [word for word in words if len(word) > 2]
  words = [word for word in words if ord(word[0]) <= 122 and (ord(word[0])
  < 48 or ord(word[0]) > 57)]
  words = [word for word in words if word[0] not in punctuation]
  words = [word for word in words if word not in english_stop_words]
  stemmer = PorterStemmer()
  words = [stemmer.stem(word) for word in words]
  for word in words:
    vector[word] += 1
  arr = list(vector.values())  # or [vector[key] for key in vector.keys()]
  return arr


def get_k_closest_documents(dictionary, normalized_idfs, data, k):
  vector = create_bag_of_words_vector(dictionary, data)
  q_vector = normalize([vector], norm="l1")[0]
  q_vector_T = np.array(q_vector).T
  arr_of_cos = normalized_idfs @ q_vector_T
  best_k_elements = [(i, arr_of_cos[i]) for i in range(len(arr_of_cos))]
  best_k_elements = sorted(best_k_elements, key=lambda x: x[1],
  reverse=True)
  return best_k_elements[:k]


query = input("Search: ")
k = 10
f = open("dictionary.json", "r")
dictionary = json.load(f)
f.close()
f = open("IDF_normalized_matrix.json", "r")
normalized_idfs = json.load(f)
f.close()
# f = open("IDF.json", "r")
# idfs = json.load(f)
# f.close()
# normalized = normalize_matrix(idfs)
# save_dictionary(normalized)
start = time.time()
documents = get_k_closest_documents(dictionary, normalized_idfs, query, k)
end = time.time()
print(documents)
print(end - start)

f = open("data.json", "r")
data = json.load(f)
f.close()
```

```
for document in documents:
  print(document, data[document[0]]["title"])
```

Powyższy kod realizuje podane polecenie. Na końcu otrzymujemy w kolejności najbardziej dopasowane dokumenty, które udało się znaleźć. Jeśli powyższy kod wywołujemy pierwsze raz należy odkomentować kod, który jest na szaro, a linijki poniżej zakomentować, gdyż wtedy utworzy nam plik, w którym będzie przechowywana macierz z unormowanymi wektorami cech.

8. Polecenie: W celu usunięcia szumu z macierzy *A* zastosuj SVD i *low rank approximation* otrzymując

$$A \approx A_k = U_k D_k V_k^T = [u_1|\dots|u_k] \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_k^T \end{bmatrix} = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

oraz nową miarę podobieństwa

$$\cos\theta_j = \frac{q^T A_k e_j}{\|q\| \|A_k e_j\|}$$

```
import numpy as np
import scipy.sparse.linalg
import string
import json
import time
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.preprocessing import normalize

english_stop_words = set(stopwords.words("english"))
punctuation = string.punctuation


def create_bag_of_words_vector(dictionary, data):
  vector = {}
  for word in dictionary.keys():
    vector[word] = 0
  text = data.lower()
  words = nltk.word_tokenize(text)
  words = [word for word in words if len(word) > 2]
  words = [word for word in words if ord(word[0]) <= 122 and (ord(word[0])
  < 48 or ord(word[0]) > 57)]
  words = [word for word in words if word[0] not in punctuation]
  words = [word for word in words if word not in english_stop_words]
  stemmer = PorterStemmer()
  words = [stemmer.stem(word) for word in words]
  for word in words:
    vector[word] += 1
  arr = list(vector.values())  # or [vector[key] for key in vector.keys()]
  return arr


def use_svd_and_low_rank_approx(normalized_idfs, k):
  u, s, vh = scipy.sparse.linalg.svds(
  scipy.sparse.linalg.aslinearoperator(np.array(normalized_idfs)), k=k)
  # u, s, vh = scipy.sparse.linalg.svds(
  scipy.sparse.linalg.aslinearoperator(np.array(normalized_idfs,
  dtype="float32")), k=k) # use this is above line doesnt work because of
```

```
  used memory
  # u, s, vh = scipy.sparse.linalg.svds(normalized_idfs, k=k) # also works
  return u @ np.diag(s) @ vh


def get_k_closest_documents(dictionary, normalized_idfs, data, k,
  approx_k):
  normalized_idfs_noise_reduced =
  use_svd_and_low_rank_approx(normalized_idfs, approx_k)
  print(len(normalized_idfs_noise_reduced),
  len(normalized_idfs_noise_reduced[0]))
  vector = create_bag_of_words_vector(dictionary, data)
  q_vector = normalize([vector], norm="l1")[0]
  q_vector_T = np.array(q_vector).T
  best_k_elements = []
  index = 0
  for document in normalized_idfs_noise_reduced:
    values = document
    cos = (q_vector_T @ np.array(values)) / (np.linalg.norm(q_vector) *
    np.linalg.norm(values))
    best_k_elements.append((index, cos))
    index += 1
    print(index)
  best_k_elements = sorted(best_k_elements, key=lambda x: x[1],
  reverse=True)
  return best_k_elements[:k]


query = input("Search: ")
k = 10
approx_k = 400
f = open("dictionary.json", "r")
dictionary = json.load(f)
f.close()
f = open("IDF_normalized_matrix.json", "r")
normalized_idfs = json.load(f)
f.close()
start = time.time()
documents = get_k_closest_documents(dictionary, normalized_idfs, query, k,
approx_k)
end = time.time()
print(documents)
print(end - start)

f = open("data.json", "r")
data = json.load(f)
f.close()
for document in documents:
  print(document, data[document[0]]["title"])
```

Wynik działania możemy zobaczyć uruchamiając powyższy kod. Jeśli macierz jest zbyt duża możemy spróbować zmienić typ przechowywanych danych przez odkomentowanie linijki, która aplikuje SVD w naszym kodzie.

9. Polecenie: Porównaj działanie programu bez usuwania szumu i z usuwaniem szumu. Dla jakiej wartości *k* wyniki wyszukiwania są najlepsze (subiektywnie). Zbadaj wpływ przekształcenia IDF na wyniki wyszukiwania.

Program bez usuwania szumu zdawał się dawać lepiej dopasowane wyniki, jednak dla stosunkowo dużych wartości *k* wynoszących około 200 otrzymywane wyniki były bardzo sensowne. Choć w wielu przypadkach brało dokumenty, w których słowo jedno ze słów występowało raz, a sam dokument był krótki.
Przekształcenie IDF znacznie zmniejsza znaczenie często powtarzających się słów w dokumentach, lecz jest wiele takich tekstów, gdzie jedno słowo powtarza się wiele razy, przez co i tak wyszukiwarka ma duże prawdopodobieństwo znalezienia takiego dokumentu.