Metody Obliczeniowe w Nauce i Technice

Laboratorium 6

Wojciech Łącki

Spis treści

[Zadanie 1 – Wyszukiwarka 2](#_Toc102503421)

# 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.

1. 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.

1. Polecenie: Dla każdego dokumentu *j* wyznacz wektor cech *bag-of-words dj* 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.

1. Polecenie: Zbuduj rzadka macierz wektorów cech *term-by-document matrix* w której wektory cech ułozone są kolumnowo *Am×n = [d1|d2| . . . |dn]* (*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.

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

gdzie *nw* 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.

1. 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

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.

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

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.

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

oraz nową miarę podobieństwa

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.

1. 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.