Untitled

April 15, 2020

1 Wyszukiwarka

[1]:
```
import os
import sys
import numpy as np
import numpy.linalg as lin
import re
from tqdm import tqdm
from collections import defaultdict
import matplotlib.pyplot as plt
```

[2]:
```
DATA_PATH = "OANC-data"
```

1.1 Przygotowanie danych

Do przetestowania wyszukiwarki dokumentów zostanie wykorzystany korpus tekstów OANC (Open American National Corpus) zawierający prawie 9000 elementów. http://www.anc.org/data/oanc/download/

[3]:
```
def move_txt(source, to):
    for name in os.listdir(source):
        if name[-3:]=="txt":
            target = to+"/"+name
            os.rename(source+"/"+name, target)
        if os.path.isdir(source+"/"+name):
            move_txt(source+"/"+name, to)
move_txt(source="data", to=DATA_PATH)
```

1.2 Określanie słów kluczowych

Jako zbiór słów kluczowych zostanie wykorzystana część zbioru będącego unią wszystkich słów występujących we wszystkich elementach korpusu. Słowa stanowiące klucze zostaną wybrane na podstawie częstotliwości ich występowania w tekście.

[3]:
```
key_words = set()
documents = [name for name in os.listdir(DATA_PATH)]#[:1500]
bag_of_words = defaultdict(lambda: defaultdict(lambda: 0))
def split(text):
    return re.findall(r"[\w]+", text)

for filename in tqdm(documents, position=0):
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    words = split(text)
    for word in words:
        bag_of_words[filename][word] += 1
    key_words |= set(words)
key_words = list(key_words)
```

```
100%|      | 8806/8806 [03:16<00:00, 44.85it/s]
```

[5]:
```
word_count = defaultdict(lambda: 0)
for doc_words in bag_of_words.values():
    for key, val in doc_words.items():
        word_count[key] += val
```

[7]:
```
len(key_words)
```

[7]:
```
163263
```

[13]:
```
print("Words used:", sum([1 for k, v in word_count.items() if v>=100]))
print("% of words discarded:",100 * sum([1 for k, v in word_count.items() if␣
 ↪v<100]) / len(key_words))
```

```
Words used: 10032
% of words discarded: 93.85531320629904
```

Jako klucze zostaną wykorzystane tylko słowa pojawiające się łącznie we wszystkich tekstach co najmniej 100 razy. Zbiór słów kluczowych liczy więc 10032 słowa, co oznacza, że odrzucone zostaje około 94% wszystkich słów. Zostaną one wszystkie zakodowane pod tym samym indeksem w wektorze bag-of-words.
Odrzucenie tak dużej ilości słów jest głównie spowodowane ograniczeniami sprzętowymi - wykonywane w dalszej części obliczenia mają duże wymagania pamięciowe, które są proporcjonalne m.in. do ilośći słów kluczowych.

[14]:
```
encoding_size = 1
word_idx = dict()
for word, count in tqdm(word_count.items(), position=0):
    if count >=100:
        word_idx[word] = encoding_size
        encoding_size += 1
    else:
```

```
        word_idx[word] = 0
```

```
100%|      | 163263/163263 [00:00<00:00, 1389226.31it/s]
```

1.3 Budowa rzadkiej macierzy wektorów cech 'term-by-document'

[16]:
```
def encode(word_idx, word_count):
    global encoding_size
    encoding = np.zeros(encoding_size, dtype=np.uint16)
    for word, count in word_count.items():
        encoding[word_idx[word]] = count
    return encoding

encodings = []
for name in tqdm(documents, position=0):
    encoding = encode(word_idx, bag_of_words[name])
    encodings.append(encoding)
```

```
100%|      | 8806/8806 [00:01<00:00, 5402.97it/s]
```

[17]:
```
term_by_document = np.stack(encodings, axis=1)
term_by_document.shape
```

[17]:
```
(10033, 8806)
```

1.4 Skalowanie macierzy cech

Każdy element macierzy cech jest mnożony przez współczynnik 'inverse document frequency' celem redukcji znaczenia często występujących słów. np. słowo występujące we wszystkich tekstach zostanie pomnożone przez 0, czyli efektywnie będzie usunięte z rozważań.

[18]:
```
def IDF(N, term_by_document):
    return np.log(N/np.sum(term_by_document != 0, axis=1, dtype=np.float32),
                  dtype=np.float32)[:, None]
N = len(documents)
term_by_document_scaled =np.multiply(term_by_document, IDF(N,␣
 ↪term_by_document), dtype=np.float32)
```

[19]:
```
term_by_document_scaled.shape
```

[19]:
```
(10033, 8806)
```

1.5 Poszukiwanie dokumentów

Funkcja `find(query, k)` wybiera k dokumentów najbardziej pasujących do zapytania `query`.

query jest napisem, który w funkcji jest rozkładany na pojedyncze słowa i kodowany do postaci wektorowej.

[22]:
```
def find(query, k):
    global term_by_document_scaled
    global documents
    global word_idx
    query_words = split(query)
    query_count = defaultdict(lambda: 0)
    for word in query_words:
        query_count[word] += 1
    encoded_query = encode(word_idx, query_count)
    probabilities = term_by_document_scaled.T @ encoded_query / (lin.
 ↪norm(encoded_query) *\
                                                          lin.
 ↪norm(term_by_document, axis=0))
    idx = np.argsort(probabilities)
   # print(max(probabilities))
  #  print(probabilities[idx[-k:]])
    return [documents[i] for i in idx[-k:]][::-1]
```

1.5.1 Przykładowe zapytania

[ ]:
```
query_res = find("water", 2)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = find("aloha united way", 2)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = find("hospital cancer water", 2)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

1.6 Normalizacja wektorów

[30]:
```
def normalise(array):
    if len(array.shape)==1:
        return array / lin.norm(array)
    else:
        return array / lin.norm(array, axis=0)
```

[31]:
```
term_by_document_normalised = normalise(term_by_document_scaled)
```

1.7 Poszukiwanie dokumentów z użyciem znormalizowanej macierzy

[32]:
```
def normalised_find(query, k):
    global term_by_document_normalised
    global documents
    global word_idx
    query_words = split(query)
    query_count = defaultdict(lambda: 0)
    for word in query_words:
        query_count[word] += 1
    encoded_query = normalise(encode(word_idx, query_count))
    probabilities = term_by_document_normalised.T @ encoded_query
    idx = np.argsort(probabilities)
   # print(max(probabilities))
  #  print(probabilities[idx[-k:]])
    return [documents[i] for i in idx[-k:]][::-1]
```

1.7.1 Przykładowe zapytania

[ ]:
```
query_res = normalised_find("water", 2)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = normalised_find("aloha united way", 2)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = normalised_find("hospital cancer water", 2)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

1.8 Usuwanie szumu przy uzyciu SVD ze znormalizowanej macierzy 'term by document'

[43]:
```
u, s, vh = lin.svd(term_by_document_normalised)
```

[44]:
```
Sigma = np.zeros((term_by_document_normalised.shape[0],␣
↪term_by_document_normalised.shape[1]))
```

[45]:
```
Sigma[:term_by_document_normalised.shape[1], :term_by_document_normalised.
↪shape[1]] = np.diag(s)
```

[46]:
```
k = term_by_document_normalised.shape[1]
```

[47]:
```
term_by_document_aprrox = u @ Sigma[:,:k] @ vh[:k,:]
```

1.9 Poszukiwanie dokumentów z użyciem odszumionej macierzy

[49]:
```
def denoised_find(query, results_num=3, k=1000):
    global u, Sigma, vh
    global documents
    global word_idx
    term_by_document_aprrox = u @ Sigma[:,:k] @ vh[:k,:]
    query_words = split(query)
    query_count = defaultdict(lambda: 0)
    for word in query_words:
        query_count[word] += 1
    encoded_query = normalise(encode(word_idx, query_count))
    probabilities = term_by_document_aprrox.T @ encoded_query
    idx = np.argsort(probabilities)
   # print(max(probabilities))
   # print(probabilities[idx[-results_num:]])
    return [documents[i] for i in idx[-results_num:]][::-1]
```

1.9.1 Przykładowe zapytania

[ ]:
```
query_res = denoised_find("aloha united way", 2, 100)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = denoised_find("aloha united way", 2, 1000)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = denoised_find("aloha united way", 2, 6000)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = denoised_find("water", 2, 6000)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

[ ]:
```
query_res = denoised_find("aloha united way", 3, 8800)
for filename in query_res:
    with open(DATA_PATH + "/" +filename, "r") as file:
        text = file.read().lower()
    print(text)
    print((("--"*40)+"\n")*3)
```

1.9.2 Wnioski dotyczące wartości k

Wartość k ma bardzo duże znaczenie przy wykorzystaniu aproksymacji macierzy 'term by document'. Subiektywnie oceniając wyniki zwracane dla niskich wartości k (do 1000) są bezwartościowe i w ogóle niezwiązane z zapytaniem. Przy k równym kilku tysiącom wyniki są praktycznie nierozróżnialne od tych uzyskanych przy użyciu nieodszumionej macierzy.

1.9.3 Wnioski dotyczące przekształcenia IDF

Osobiście nie mogę stwierdzić, żeby wyniki zapytań znacząco się różniły w zależności od modyfikacji macierzy 'term by document'. Wyniki, które uzyskałem przy niezmodyfikowanej macierzy i macierzy przekształconej przez IDF są na tyle zbliżone, że różnica nie jest jednoznacznie możliwa do ocenienia subiektywnie.