Metody obliczeniowe w nauce i technice 2

Tomasz Zawadzki

29 maja 2019

Zadanie 1. Wyszukiwarka

Zbiorem dokumentów tekstowych jest zrzut Wikipedii Simple English (`simple.wikipedia.org`) dostępny pod adresem `https://github.com/LGDoor/Dump-of-Simple-English-Wiki`. Zawiera on 50441 artykułów w języku angielskim.

Treść artykułów jest dzielona na tokeny przy użyciu funkcji `word_tokenize` z pakietu do przetwarzania języka naturalnego `nltk`. Uwzględniane są jedynie termy alfanumeryczne o długości co najmniej 3 znaków, które nie są tzw. *stop words*. Termy są następnie stemowane przy użyciu algorytmu Porter Stemmer (`nltk.stem.PorterStemmer`).

```
self.translate_table = str.maketrans('', '', string.punctuation)
self.stemmer = nltk.stem.PorterStemmer()
self.stopwords = nltk.corpus.stopwords.words('english')
return [
    self.stemmer.stem(token)
    for token in nltk.word_tokenize(content.translate(self.translate_table))
    if len(token) >= 3 and token.isalnum() and token not in self.stopwords
]
```

Słownikiem słów kluczowych jest unia wszystkich termów występujących we wszystkich tekstach oraz tytułach artykułów. Enumeracja tego zbioru stanowi bijekcję z indeksami wierszy macierzy *term-by-document*. W analogiczny sposób numerowane są dokumenty.

```
documents = dict(enumerate(self.source.documents()))
terms = set()
for name, _, content in documents.values():
    terms |= set(self.termifier.termify(name + ' ' + content))
terms = {v: k for k, v in enumerate(list(terms))}
```

Na podstawie liczb wystąpień (ang. *term frequency*) termów w poszczególnych dokumentach, zliczanych przy użyciu wbudowanej kolekcji `collections.Counter`, uzupełniana jest rzadka macierz *term-by-document*.

$$a_{i,j} = \mathrm{tf}(i,j) = |\{t \in D_j : t = t_i\}| \tag{1}$$

$$\begin{array}{c} \\ \mathbf{t}_i \rightarrow \end{array} \begin{array}{c} \mathbf{d}_j \\ \downarrow \\ \begin{bmatrix} a_{1,1} & \dots & a_{1,j} & \dots & a_{1,|D|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i,1} & \dots & a_{i,j} & \dots & a_{i,|D|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{|T|,1} & \dots & a_{|T|,j} & \dots & a_{|T|,|D|} \end{bmatrix} \end{array} \tag{2}$$

```
matrix = scipy.sparse.lil_matrix((len(terms), len(documents)))
for document_id, (name, url, content) in documents.items():
    for term_id, tf in collections.Counter(
        [terms[term] for term in self.termifier.termify(name + ' ' + content)]
    ).items():
        matrix[term_id, document_id] = tf
```

Następnie dla każdego termu zostaje obliczona *Inverse Document Frequency* (IDF) w postaci wektora kolumnowego, którego i-tą współrzędną jest logarytm z liczby wszystkich dokumentów podzielonej przez liczbę dokumentów zawierających przynajmniej jedno wystąpienie i-tego termu (czyli liczbę niezerowych elementów w i-tym wierszu *term-by-document matrix*).

$$\operatorname{idf}(i) = \log \frac{|D|}{|\{d \in D : t_i \in d\}|} \tag{3}$$

$$\operatorname{IDF} = [\, \operatorname{idf}(1) \mid \operatorname{idf}(2) \mid \dots \mid \operatorname{idf}(|T|) \,]^{\mathrm{T}} \tag{4}$$

```
occurrences = scipy.sparse.linalg.norm(matrix, ord=0, axis=1)
idf = numpy.log(float(len(documents)) *
          numpy.reciprocal(occurrences, dtype=numpy.float64))
```

Kolumny macierzy *term-by-document* są nastepnie mnożone przez wektor IDF

$$a_{i,j} = \text{tf–idf}(i,j) = \operatorname{tf}(i,j) \cdot \operatorname{idf}(i), \tag{5}$$

a następnie normalizowane według normy euklidesowej

$$\|\mathbf{d}_j\| = \sqrt{\sum_{i=1}^{|T|} {a_{i,j}}^2} = 1, \tag{6}$$

co pozwala bardziej efektycznie obliczać współczynniki korelacji pomiędzy dokumentami.

```
matrix = matrix.T.multiply(idf).T
matrix = sklearn.preprocessing.normalize(matrix, axis=0, norm='l2', copy=False)
```

Powstała w ten sposób rzadka macierz *term-by-document* TF-IDF jest zapisywana do pliku.

Macierz jest następnie aproksymowana macierzą niższego rzędu (*low-rank approximation*)

$$\mathbf{A} \cong \mathbf{A}_k, \quad \operatorname{rank}(\mathbf{M}_k) \leqslant k \ll \min\{|T|, |D|\}, \tag{7}$$

wykorzystując rozkład według wartości osobliwych (*Singular value decomposition*, SVD) przy użyciu `TruncatedSVD` z pakietu `sklearn.decomposition`.

$$\mathbf{A} \cong \mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^{\mathrm{T}}. \tag{8}$$

Macierz $\mathbf{A}_k$ jest skompresowanym, pozbawionym szumu, najlepszym przybliżeniem $\mathbf{A}$ (według normy), które można osiągnąć za pomocą macierzy rzędu k.

```
svd = sklearn.decomposition.TruncatedSVD(n_components=self.svd_k).fit(matrix.T)
svd_matrix = svd.transform(matrix.T)
svd_components = svd.components_
```

Współczynnik korelacji pewnego zapytania z j-tym dokumentem można wyrazić wzorem

$$\cos\theta_j = \hat{\mathbf{q}} \circ \hat{\mathbf{d}}_j = \frac{\mathbf{q}}{\|\mathbf{q}\|} \circ \frac{\mathbf{d}_j}{\|\mathbf{d}_j\|} = \frac{\mathbf{q} \circ \mathbf{d}_j}{\|\mathbf{q}\| \, \|\mathbf{d}_j\|} = \frac{\mathbf{q}^{\mathrm{T}} \mathbf{d}_j}{\|\mathbf{q}\| \, \|\mathbf{d}_j\|} = \mathbf{q}^{\mathrm{T}} \mathbf{d}_j \tag{9}$$

co pozwala obliczyć wektor korelacji zapytania z każdym z dokumentów jako $\mathbf{q}^{\mathrm{T}}\mathbf{A}$, gdzie $\mathbf{q}$ jest znormalizowanym wektorem wystąpień termów w zapytaniu, a $\mathbf{A}$ jest macierzą *term-by-document*.

```
q = scipy.sparse.lil_matrix((len(self.terms), 1))
term_ids = list(set([
    self.terms[term]
    for term in set(self.termifier.termify(query))
    if term in self.terms
]))
if len(term_ids) == 0:
    return {'error': 'no_query'}
q[term_ids,0] = 1.0 / math.sqrt(len(term_ids))
```

Wektor zapytania jest transformowany do przestrzeni niższego wymiaru k, w której obliczana jest jego korelacja z macierzą $\mathbf{A}_k$, stanowiącej przybliżenie macierzy *term-by-document* $\mathbf{A}$ w przestrzeni k-wymiarowej. Wektor korelacji w przestrzeni pełno-wymiarowej jest wynikiem odwrotnej transformacji wektora korelacji w przestrzeni niższego wymiaru. Wyniki wyszukiwania są sortowane nierosnąco względem obliczonego współczynnika korelacji.

```
svd_q = self.svd_components.dot(q.todense())
svd_c = self.svd_matrix.dot(svd_q)
correlations = {
    document_id: svd_c[document_id,0]
    for document_id in range(len(self.documents))
}
results = sorted(correlations.items(), key=itemgetter(1), reverse=True)
```

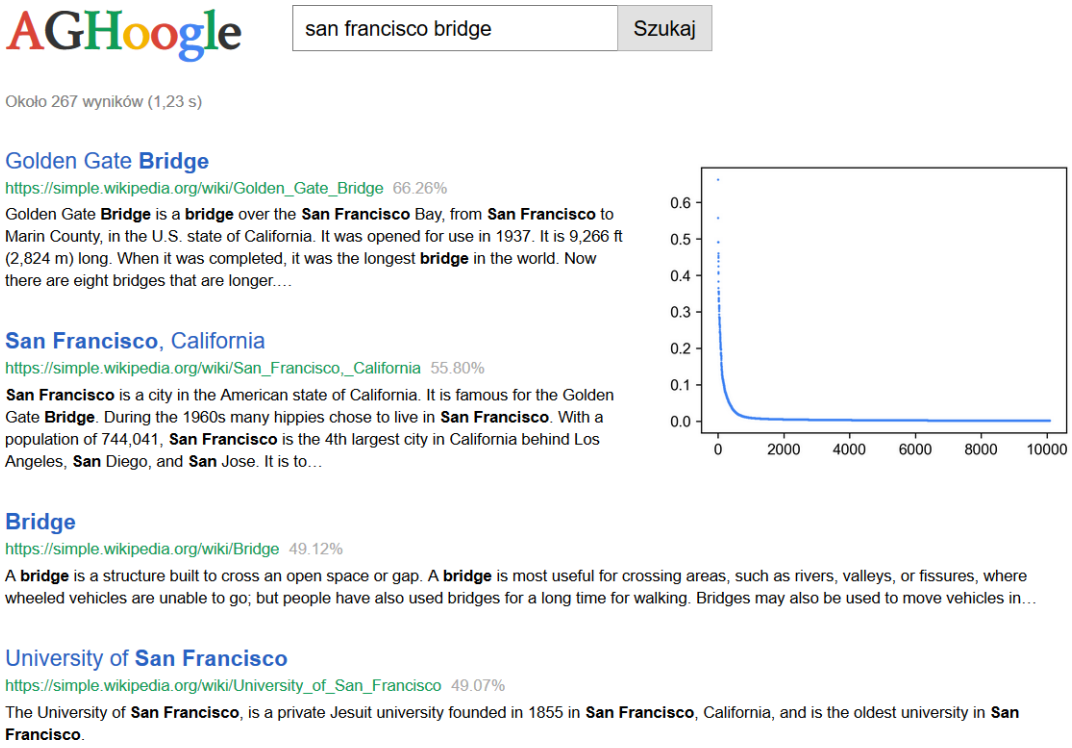
Ponieważ każdy wymiar przestrzeni pełnowymiarowej jest związany z dokładnie jednym termem, to analogicznie można interpretować kolejne wymiary przestrzeni niższego wymiaru jako różne tematy rozumiane jako podzbiory termów. Poniższa lista zawiera najciekawsze połączenia zidentyfikowane przez dla $k = 1000$.

4. depart; franc; commun; region; found
5. championship; overview; team; leagu; statist
7. illinoi; oklahoma; florida; counti; citi
15. district; canton; switzerland; calvado; département
18. game; footbal; play; player; leagu
19. class; rail; british; locomot; built
55. number; day; calendar; movi; gregorian
63. moon; book; jupit; alabama; saturn
67. london; olymp; award; borough; contest;

W analogiczny sposób grupowane są dokumenty. Poniższa lista zawiera najciekawsze połączenia zidentyfikowane przez dla $k = 1000$.

68. Mu (letter); Nu (letter); Pi (letter); Xi; San (letter)
74. Lake Zürich; Lake; Great Lakes; Crater Lake; Lake Constance
103. Cell; Cell growth; Cell theory; Multicellular organism; T cell
105. Law school; High school; Middle school; School; Boarding school
133. Windows 95; Windows 2000; Mountain; Windows 2.0; Windows Me
144. Key generation; Symmetric-key algorithm; Key exchange; Key size; Key space
151. Key space; Key exchange; Public-key cryptography; Symmetric-key algorithm; Key generation
162. Oil painting; Paint; Fresco; Art; Sistine Chapel
170. Month; Calendar; Natural satellite; Islamic calendar; Chinese calendar
207. Super Mario Bros.; New Super Mario Bros.; Super Mario 64; Mario (Nintendo); Super Mario Bros. 3
217. Windows Me; Windows 98; Windows 2.0; Windows Defender; Windows 95
289. Castle (disambiguation); Castle; Windsor Castle; Carisbrooke Castle; Neuschwanstein Castle
377. Simple English Wikipedia; Arabic Wikipedia; Wu Wikipedia; Wikipedia; Nupedia
384. Tokyo Stock Exchange; Stock exchange; London Stock Exchange; Bombay Stock Exchange; New York Stock Exchange
550. Scrooge McDuck; The Richest Duck in the World; King of the Klondike; The Master of the Mississippi; Ducks of the West
608. Bengal tiger; Tiger; Bali tiger; Indo-Chinese tiger; Bangladesh
613. Apple; Bengal tiger; Tiger; Mac OS X v10.4; Apple Corps
644. Mineral; Mineral exploration; Crystal Palace; List of minerals; Salt
853. University of Houston; Houston Rockets; Houston, Texas; Houston Astros; 3031 Houston
877. Cuba; Cuban Missile Crisis; Fidel Castro; Santiago de Cuba province; Cuban Missile Crisis
901. Elephant; African Buffalo; African elephant; African Bush Elephant; Asian Elephant
902. King Penguin; Gentoo Penguin; Club Penguin; Penguin; March of the Penguins
920. Comet; List of comets; Comet Halley; Comet Borrelly; Comet West;

Rys. 1. Ekran powitalny wyszukiwarki AGHoogle



Rys. 2. Wyniki wyszukiwania dla zapytania "san francisco bridge". Wykres po prawej stronie prezentuje wartości korelacji dla pierwszych 10000 znalezionych rezultatów wyszukiwania

Bibliografia

[1] https://en.wikipedia.org/wiki/Latent_semantic_analysis

[2] https://en.wikipedia.org/wiki/Document-term_matrix

[3] https://en.wikipedia.org/wiki/tf-idf

[4] https://en.wikipedia.org/wiki/Low-rank_approximation

[5] https://en.wikipedia.org/wiki/Singular_value_decomposition

[6] https://docs.scipy.org/doc/scipy/reference/sparse.html

[7] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html