МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА
Факультет информатики и систем управления
Кафедра теоретической информатики и компьютерных технологий

Лабораторная работа №17
по курсу «Информационный поиск»

«EM-алгоритм»

Выполнил:
студент группы ИУ9-21М
Беляев А. В.

Проверила:
Лукашевич Н. В.

Москва 2019

1 Цель работы

Подсчитать вероятности отнесения слов и документов к темам за 5 итераций EM-алгоритма (Expectation maximization).

1.1 Ход работы

```python
from math import sqrt
import numpy as np

np.set_printoptions(precision=3)

DICT = ['w0', 'w1', 'w2']
DOCS = [['w0', 'w1', 'w1'],
        ['w0', 'w1', 'w2'],
        ['w0', 'w2', 'w2']]
ITERS = 5
THEMES = 2

def rand_init_matrix(cols: int, rows: int) -> list:
    matrix = np.random.rand(rows, cols)
    m_sum = sum(np.sum(matrix, 0).tolist())
    matrix /= m_sum

    matrix = matrix.transpose().tolist()
    for i in range(len(matrix)):
        s = sum(matrix[i])
        for j in range(len(matrix[i])):
            matrix[i][j] /= s
    return np.array(matrix).transpose().tolist()


def em_step(m_words: list, m_docs: list, docs: list):
    def n_dwt(thm_id: int, word_id: int, doc_id: int) -> float:
        p = m_words[word_id][thm_id] * m_docs[thm_id][doc_id]
        s = 0
        for i in range(THEMES):
            s += m_words[word_id][i] * m_docs[i][doc_id]
        return docs[doc_id].count(DICT[word_id]) * (p / s)

    def n_wt(thm_id: int, word_id: int) -> float:
        s = 0
        for i in range(len(docs)):
            s += n_dwt(thm_id, word_id, i)
        return s

```

```
    def n_td(thm_id: int, doc_id: int) -> float:
        s = 0
        doc = docs[doc_id]
        for i in range(len(doc)):
            s += n_dwt(thm_id, i, doc_id)
        return s

    def n_t(thm_id: int) -> float:
        s = 0
        for i in range(len(DICT)):
            s += n_wt(thm_id, i)
        return s

    def n_d(doc_id: int) -> float:
        s = 0
        for i in range(THEMES):
            s += n_td(i, doc_id)
        return s

    m_fi = np.zeros((len(docs), THEMES))
    for i in range(len(docs)):
        for j in range(THEMES):
            m_fi[i][j] = n_wt(j, i) / n_t(j)

    m_psi = np.zeros((THEMES, len(DICT)))
    for i in range(THEMES):
        for j in range(len(DICT)):
            m_psi[i][j] = n_td(i, j) / n_d(j)
    return m_fi, m_psi


def main():
    m_words = rand_init_matrix(THEMES, len(DICT))
    m_docs = rand_init_matrix(len(DOCS), THEMES)

    print(f'words:\n{np.array(m_words)}')
    print(f'docs:\n{np.array(m_docs)}')

    for i in range(ITERS):
        m_words, m_docs = em_step(m_words, m_docs, DOCS)
        print(f'{i}: words:\n{np.array(m_words)}')
        print(f'{i}: docs:\n{np.array(m_docs)}')

if __name__ == '__main__':
    main()
```

2 Результаты

На листинге ниже приведены исходные матрицы и матрицы, полученные на соответствующих итерациях алгоритма $[0, 4]$. распределени по двум темам:

```
words:
[[0.244 0.361]
 [0.425 0.29 ]
 [0.331 0.349]]
docs:
[[0.301 0.596 0.913]
 [0.699 0.404 0.087]]

0: words:
[[0.293 0.395]
 [0.267 0.435]
 [0.44  0.169]]
0: docs:
[[0.334 0.589 0.898]
 [0.666 0.411 0.102]]

1: words:
[[0.312 0.363]
 [0.177 0.557]
 [0.511 0.08 ]]
1: docs:
[[0.247 0.59  0.928]
 [0.753 0.41  0.072]]

2: words:
[[0.333 0.333]
 [0.099 0.635]
 [0.568 0.031]]
2: docs:
[[0.136 0.59  0.964]
 [0.864 0.41  0.036]]

3: words:
[[0.346 0.318]
 [0.047 0.672]
 [0.606 0.01 ]]
3: docs:
[[0.061 0.579 0.987]
 [0.939 0.421 0.013]]

4: words:
```

```
[[0.349 0.316]
 [0.021 0.681]
 [0.63  0.003]]
4: docs:
[[0.025 0.559 0.996]
 [0.975 0.441 0.004]]
```

Распеределение по трем темам:

```
words:
[[0.1987 0.2557 0.2839]
 [0.4721 0.0767 0.209 ]
 [0.3292 0.6675 0.5071]]
docs:
[[0.0703 0.2989 0.3655]
 [0.7303 0.4927 0.1996]
 [0.1995 0.2084 0.435 ]]

0: words:
[[0.2438 0.3963 0.329 ]
 [0.4707 0.2809 0.2849]
 [0.2855 0.3228 0.3861]]
0: docs:
[[0.187  0.3539 0.2671]
 [0.5273 0.4341 0.2562]
 [0.2857 0.212  0.4767]]

1: words:
[[0.2496 0.3954 0.3246]
 [0.4328 0.3527 0.2302]
 [0.3177 0.2519 0.4452]]
1: docs:
[[0.2284 0.3511 0.2153]
 [0.5108 0.4359 0.2653]
 [0.2608 0.213  0.5193]]

2: words:
[[0.2555 0.3934 0.3211]
 [0.4334 0.411  0.1627]
 [0.3111 0.1956 0.5162]]
2: docs:
[[0.2497 0.3491 0.1792]
 [0.5502 0.4348 0.2285]
 [0.2001 0.2161 0.5923]]

```

```
3: words:
[[0.2673 0.3905 0.3138]
 [0.4502 0.4726 0.0876]
 [0.2825 0.1369 0.5986]]
3: docs:
[[0.2584 0.3469 0.1383]
 [0.6203 0.429  0.1653]
 [0.1214 0.2241 0.6964]]

4: words:
[[0.2882 0.3862 0.3034]
 [0.469  0.5321 0.031 ]
 [0.2428 0.0817 0.6656]]
4: docs:
[[0.2505 0.3433 0.0929]
 [0.6963 0.4147 0.0988]
 [0.0532 0.2419 0.8083]]
```

3 Выводы

Были подсчитаны вероятности отнесения документов и слов к темам алгоритмом EM.

В зависимости от начальных матриц, алгоритм «уверенно относит» перый и последний документы к разным темам (1 и 2 или наоборот). С отнесением второго документа алгоритм «не может» определиться.

Похожая ситуация со словами – второе и третье слова достаточно уверенно отнесены к темам, а с отнесением первого слова алгоритм вновь не оперделился.

С распередленеем по трем темам похожая ситуация, но решения лагоритма чуть менее «уверенные».