Information Retrieval

WS 2017 / 2018

Lecture 8, Tuesday December 12nd, 2017
(Vector Space Model)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

Overview of this lecture

- Organizational
  - Your experiences with ES7 — Web app, part 2
  - Demo of some web apps
- Contents
  - Encoding — last part of L7
  - Vector Space Model (VSM) — documents as vectors
  - ES8: re-implement your code from ES2 using the VSM, and re-evaluate benchmark

Experiences with ES7 1/2

- Summary / excerpts
  - Again, many of you liked this **a lot**
  - Less work than ES6
  - Most time spent on encoding issues (harder in C++), perfecting the design (optional), or playing Gorillas
  - "After 5 hours, we still haven't tried all gravitation settings"
  - "The last sub-task was hard to implement, because we think it is morally wrong to remove these easter eggs"
  - "Fit of rage, because index building takes so long (Java)"

Experiences with ES7 2/2

- Demos
  - Many of you produced some really nice web apps

    Let's look at a small selection together

    (and maybe a few more next week)
  - Let us also appreciate the easter (or rather xmas) eggs that were hidden in wikidata-entities-with-surprises.tsv and emerged for (not only) these queries:

    the mätrix, nirwana, gorilas, harlem, mikrosöft, snow, turn around, asteroids

Vector Space Model 1/9

- Motivation
  - For this lecture, it will be useful to represent documents as **vectors** ... here is our running example for today:

	D_1	D_2	D_3	D_4	D_5	D_6
internet	1	1	0	1	0	0
web	1	0	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

  - Each row corresponds to a word, each column to a document
  - Non-zero entries: score for that word in that document

  In the lecture, we use tf scores ... for ES8, use BM25 scores

Vector Space Model 2/9

- Terminology
  - Often referred to as the **Vector Space Model (VSM)**
  - In the VSM, words are traditionally referred to as **terms**
  - Putting the vectors from all documents from a given corpus side by side gives us the so-called **term-document matrix**

	D_1	D_2	D_3	D_4	D_5	D_6
internet	1	1	0	1	0	0
web	1	0	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Vector Space Model 3/9

Q = web surfing

- Retrieval
  - A query can also be represented as a vector ... we take 1 for a term used in the query, and 0 for all other terms
  - We measure the relevance of a document to the query by taking the **dot product** of the two vectors

    Note: this is exactly the same score as in Lecture 2

	D_1	D_2	D_3	D_4	D_5	D_6	Q
internet	1	1	0	1	0	0	0
web	1	0	1	1	0	0	1
surfing	1	1	1	2	1	1	1
beach	0	0	0	1	1	1	0
$D_i \bullet Q$	2	1	2	3	1	1	

dot product

Vector Space Model 4/9

$$\underbrace{(0,1,1,0)}_{q^T \quad 1\times 4} \cdot \overset{A \quad 4\times 6}{\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}} = \overset{1\times 6}{(2,1,2,3,1,1)}$$

- Algebra
  - More formally, let us write A for the term-document matrix and q for the query vector
  - Then the matrix-vector product $q^T \cdot A$ gives us a vector with the relevance scores of all the documents

  Let us implement this together now

	D_1	D_2	D_3	D_4	D_5	D_6		Q
internet	1	1	0	1	0	0		0
web	1	0	1	1	0	0		1
surfing	1	1	1	2	1	1		1
beach	0	0	0	1	1	1		0
				A				q

Vector Space Model 5/9

- Basic linear algebra in Python
  - For standard linear algebra, we can use **numpy**

    ```
    sudo apt-get install python3-numpy
    ```

    ```
    import numpy
    A = numpy.array([[1, 1, 0, 1, 0, 0], ...])
    q = numpy.array([0, 1, 1, 0])
    scores = q.dot(A)
    print(scores)
    ```

    Use **numpy.array** and **dot** for multiplication, not *

    q is a row vector above = q^T from the previous slide
  - See the code from the lecture for more example usage

Vector Space Model 6/9

- Sparse matrices
  - Most entries in a term-document matrix are **zero**

    Storing all entries explicitly is infeasible for large matrices

  - Sparse-matrix representation: store only the non-zero entries (together with their row and column index)

(**1**, 0, 0), (**1**, 0, 1), (**1**, 0, 3), ..., (**2**, 2, 3), ...

		0	1	2	3	4	5
		D_1	**D_2**	**D_3**	**D_4**	**D_5**	**D_6**
0	internet	**1**	**1**	0	**1**	0	0
1	web	1	0	1	1	0	0
2	surfing	1	1	1	**2**	1	1
3	beach	0	0	0	1	1	1

Vector Space Model 7/9

- Sparse matrices
  - Two principle ways to store the list of non-zero values

    row-major: store row by row (sort by row index first)

    column-major: store col by col (sort by col index first)

  - Note: the sparse row-major representation of a term-document matrix is equivalent to an **inverted index**

    (1, 0, 0), (1, 0, 1), (1, 0, 3) ids of docs containing term 0
    (1, 1, 0), (1, 1, 2), (1, 1, 3) ids of docs containing term 1
    (1, 2, 0), (1, 2, 1), (1, 2, 2) ... ids of docs containing term 2
    (1, 3, 3), (1, 3, 4), (1, 3, 5) ids of docs containing term 3

    (non-zero score, row index = term id, col index = doc id)

Vector Space Model 8/9

- Normalization
  - The idf part from BM25 or tf.idf can be seen as a **normalization** of the term document matrix:

    multiply each row by a certain factor (the idf)

  - Another typical normalization for matrices is such that the rows or columns have norm 1

    L1-norm: sum of the absolutes of the entries

    L2-norm: sum of the squares of the entries

    For ES8, play around with different normalizations of the TD matrix and see whether it improves the results

Vector Space Model 9/9

- Sparse matrices in Python
  - Not included in numpy, we have to use **scipy**

```
sudo apt-get install python3-scipy
```

```
import scipy.sparse
nz_vals   = [1, 1, 1, 1, 1, 1, ...]
row_inds = [0, 0, 0, 1, 1, 1, ...]
col_inds  = [0, 1, 3, 0, 2, 3, ...]
A = scipy.sparse.csr_matrix((nz_vals, (row_inds, col_inds)))
q = scipy.sparse.csr_matrix([0, 1, 1, 0])
scores = q.dot(A)
print(scores)
```

"csr" stands for "compressed sparse row"

References

- Textbook

  Section 6.3: The vector space model for scoring

- Linear algebra in Python
  - http://www.numpy.org
  - http://www.scipy.org
  - You find a Python numpy/scipy cheat sheet on the **wiki**