Multimedia Technology

Lecture 3 Document Retrieval: searching

Lecturer: *Dr*. Wan-Lei Zhao

Autumn Semester 2022

Email: wlzhao@xmu.edu.cn,

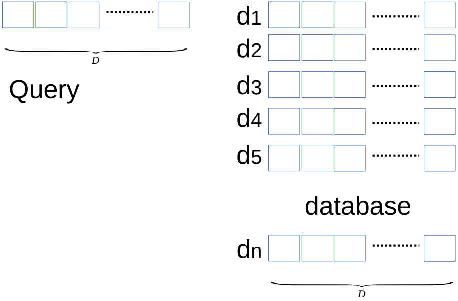
Outline

Recap

- Documents are undergone a series pre-processing
  - Tokenization, stemming and etc.
  - Only index terms are left
- Boolean model, vector model and probabilistic model
  - Vector model is selected for its simplicity and good performance
- Ready to do the retrieval
- Brute-force comparison between query and all documents in the database is not efficient
  - Dimension of query is high ($> 10,000$), size of database is in billion level
  - You cannot expect users are all as patient as you:)
  - Problem of nearest neighbour search (NNS) in high-dimensional and sparse space

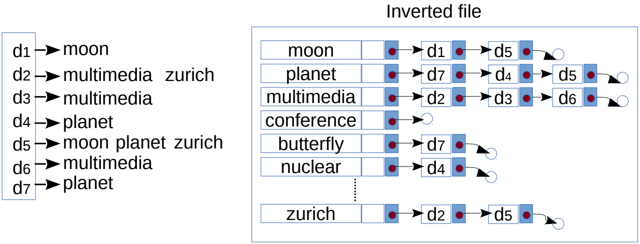
Brute-force search: illustration

- NNS is a very challenging problem, whose complexity is $\mathcal{O}(D * n)$
- Fortunately, the vectors are all sparse!!



Inverted file construction

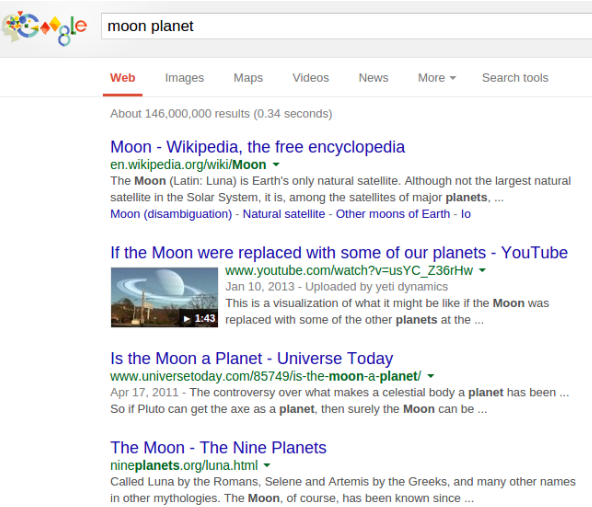
- Idea: term points towards which documents it occurs



- Two representations keep the same amount of information
- Given a query "moon planet", inverted lists pointed by "moon" and "planet" will be considered
- This way is more efficient

A complete example of inverted file (1)

- Check carefully what we still miss as database of a search engine
- Matched key words are all in **bold**



A complete example of inverted file (2)

- How we can achieve that?
- "...keyword..."
- We should know where to insert ""
- This is achieved by keeping the position of one word in the original documents
- Notice that document can be viewed as a linear array of words
- The start and rear positions of one word are kept

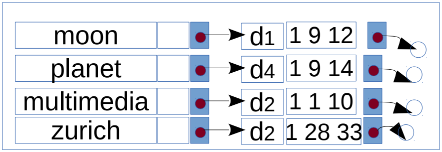
A complete example of inverted file (3)

- Sometimes, the original document is segmented into blocks
- The block ID instead of the word position is kept
- Linear search (KMP) is required to localize the word in the block

A complete example of inverted file (4)

- For one cell in the inverted list, we should keep
  - position of one word in the original document, two integers
  - document ID (integer)
  - term frequency (TF), (integer/float)
  - URLs for each document
  - Click-in frequency
  - Pagerank

Inverted file



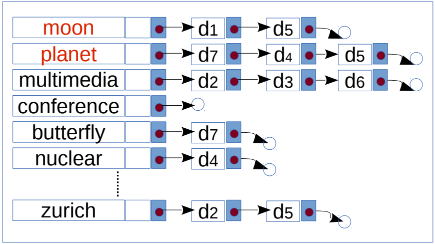
Issues involved in retrieval over inverted files (1)

- The database is now ready
- Given query "moon planet",
- Shall we compare "moon" with each inverted list leading term?
  - This is a linear scan (time consuming)
  - Any more convenient way??

query | moon | planet

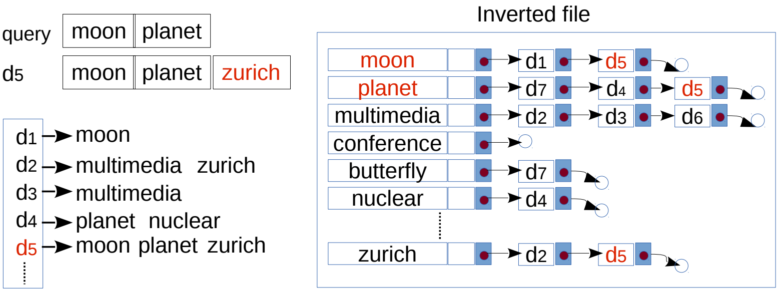
```
for each term in inverted files
    M = strcmp("moon", term)
    if (M)
      break;
end
```

Inverted file



Issues involved in retrieval over inverted files (2)

- How we compute the Euclidean distance between query and docs in the inverted file?
- Notice that "zurich" inverted list will not be considered



Outline

Retrieval on Inverted files (1)

- We have now mapped query terms into IDs
- Able to check all matched inverted lists
- How distance between query and docs from database is calculated?
- e.g. ℓ_2

$$d(q, d_j) = \sqrt{\sum_i (w_{iq} - w_{ij})^2} \quad (1)$$

- e.g. *Cosine* distance

$$cos(q, d_j) = \frac{\sum_i w_{iq} \cdot w_{ij}}{\sqrt{\sum_i w_{iq}^2} \cdot \sqrt{\sum_i w_{ij}^2}} \quad (2)$$

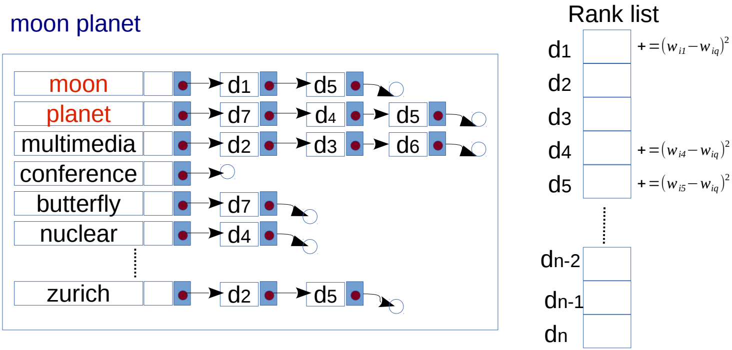
Retrieval on Inverted files (2)

- Problem: Not all w_{ij}s will be included
- If word i does not appear in the query, w_{ij} is missing
- How distance between query and docs from database is calculated?

$$d(q, d_j) = \sqrt{\sum_i (w_{iq} - w_{ij})^2} \tag{3}$$

Retrieval on Inverted files (3)

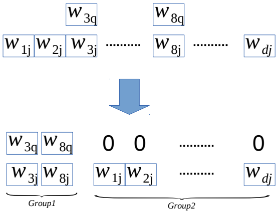
- Maintain a rank list in the memory
- Update entries according to visited inverted lists



Retrieval on Inverted files (4)

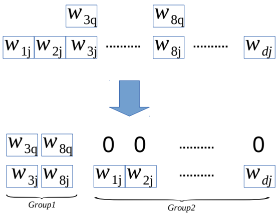
- Solution: a rank list is kept in the memory
- on which we can update from time to time
- w_{ij}s are partitioned into two groups
- Group1: those appear in the query; Group2: those do not

$$d(q, d_j) = \sqrt{\sum_i (w_{iq} - w_{ij})^2} \tag{4}$$



Retrieval on Inverted files (5)

- Solution: a rank list is kept in the memory
- on which we can update from time to time
- w_{ij}s are partitioned into two groups
- Group1 (G1): those appear in the query; Group2 (G2): those do not

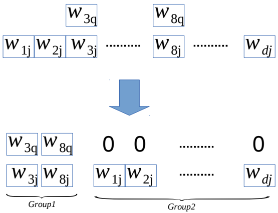


$$d^2(q, d_j) = \sum_{w_i \in G1} (w_{iq} - w_{ij})^2 + W_j - \sum_{w_i \in G1} w_{ij}^2, \qquad (5)$$

where $W_j = \sum_i w_{ij}^2$

Retrieval on Inverted files (5)

- In *Cosine* distance
- Only terms that co-occur in the query and docs are considered
- $\sqrt{\sum_i w_{ij}^2}$ is pre-computed
- $\sqrt{\sum_i w_{iq}^2}$ is computed online

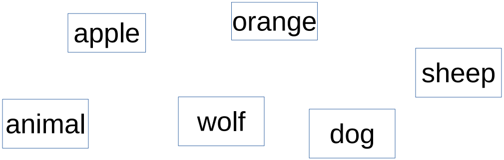


$$cos(q, d_j) = \frac{\sum_i w_{iq} \cdot w_{ij}}{\sqrt{\sum_i w_{iq}^2} \cdot \sqrt{\sum_i w_{ij}^2}} \tag{6}$$

Outline

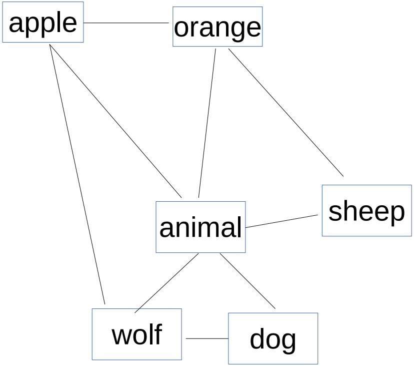
Overview on Word Embedding (1)

- In some scenario, we may need to evaluate the distance between words?
- For instance, how “apple” is similar to “orange”, in contrast to “dog”



- We are looking for a vector representation for a word
- Based on which the semantic distance between different words could be measured

Overview on Word Embedding (2)



- We are looking for a vector representation for a word
- Based on which the semantic distance between different words could be measured

Review on Bag-of-Word

	D1	D2	D3	......	Dn
apple	1	1	1		1
animal		1	1		1
dog					3
				......	
orange	2		2		
sheep		1			1
wolf					4

- We see above model before
- Each document is represented by frequencies of words

Word Embedding by Bag-of-Word

	D1	D2	D3	......	Dn
apple	1	1	1		1
animal		1	1		1
dog					3
orange	2		2	......	
sheep		1			1
wolf					4

- If we look at each row
- Each word is represented by *n* documents
- This way actually works!!

Motivation of Co-occurrence Matrix

- Besides Bag-of-Word model, there is another popular way
- It considers the co-occurrence rate of word appear in the same context

John likes <u>apple</u> and <u>orange</u>

Tom likes <u>banna</u> and <u>cherries</u>

- "apple" and "orange" play similar roles in the sentences of the same structure
- We have good reason to guess that they are something semantically similar or parallel
- "Co-occurrence" means "occur together"

Co-occurrence Matrix (1)

john like apple orange. john like football basketball.

Vocab.	john	like	apple	orange	football	basketball
john	0	2	0	0	0	0
like	2	0	1	0	1	0
apple	0	1	0	1	0	0
orange	0	0	0	0	0	0
football	0	1	0	0	0	1
basketball	0	0	0	0	1	0

- There is one mistake, please point it out for me:)

Co-occurrence Matrix (2)

john like apple orange. john like football basketball.

Vocab.	john	like	apple	orange	football	basketball
john	0	2	0	0	0	0
like	2	0	1	0	1	0
apple	0	1	0	1	0	0
orange	0	0	1	0	0	0
football	0	1	0	0	0	1
basketball	0	0	0	0	1	0

1. The matrix is symmetric
2. The diagonal elements are all zeros
3. It is sparse!!

From Co-occurrence Matrix to Word Embedding (1)

$$\begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \Longrightarrow \boxed{\begin{array}{c}\text{SVD} \\ \text{or} \\ \text{PCA}\end{array}} \xrightarrow{\text{Major components}} \begin{bmatrix} 0.51 & 0.212 \\ 0.332 & 0.517 \\ 0.214 & 0.71 \end{bmatrix}$$

- Each row in the resulting matrix gives vector representation of one word
- This is the conventional way of word embedding

From Co-occurrence Matrix to Word Embedding (2)

$$C = \begin{bmatrix} 0 & 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\Downarrow$$

$$[U \quad S \quad V] = svd(C)$$

- Given $U'_{n\times k}$ takes the first k columns of U, $S'_{k\times k}$ takes the first k rows and columns of S
- Each row of F is the vector representation for a corresponding word

$$F = U'_{n\times k} \times S'_{k\times k} \quad (7)$$

From Co-occurrence Matrix to Word Embedding (3)

$$[U \quad S \quad V] = svd(C)$$

- Given $U'_{n\times k}$ takes the first k columns of U, $S'_{k\times k}$ takes the first k rows and columns of S
- Each row of F is the vector representation for a corresponding word

$$F = U'_{n\times k} \times S'_{k\times k}$$
$$d(i,j) = cos(F(i,:), F(j,:)) \tag{8}$$

- We are now able to calculate the distance (usually *Cosine* distance) between words
- The larger the corpus is, the more it makes sense

Window Size for the Co-occurence Matrix

- We can actually set the window size with different widths

john like $\boxed{\text{apple orange}}$. jonh like football basketball.

- When width equals to 3, we study of the co-occurrence of consecutive two words in the window

john $\boxed{\text{like apple orange}}$. jonh like football basketball.

Other Approaches for Word Embedding

- Recently, deep learing approaches based on Recurrent Neural Network is quite popular
  1. Continuous Bag-of-Word: $P(word|context)$
  2. Skip-Gram: $P(context|word)$

Q & A

Thanks for your attention!