VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

DS307

SOCIAL MEDIA ANALYSIS

Faculty of Information Science and Engineering

University of Information Technology, VNU-HCM

This Course's Contents

Document Retrieval

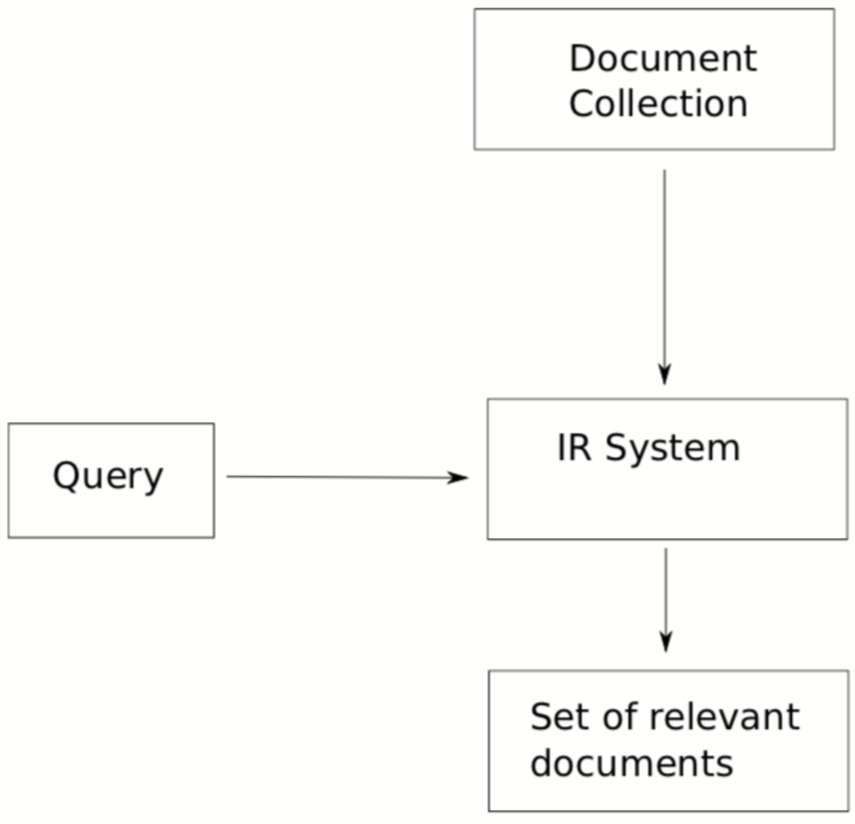
Document Retrieval

- Retrieve documents with content that is relevant to a user's information need
- Document set is fixed (size can vary from 10s of documents to billions)
- Information need is not fixed (ad-hoc retrieval)

Goal:

- Documents relevant to query should be returned
- Documents not relevant to query should not be returned

Document Retrieval Architecture



Documents as vectors

- So we have a |V|-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.

Queries as vectors

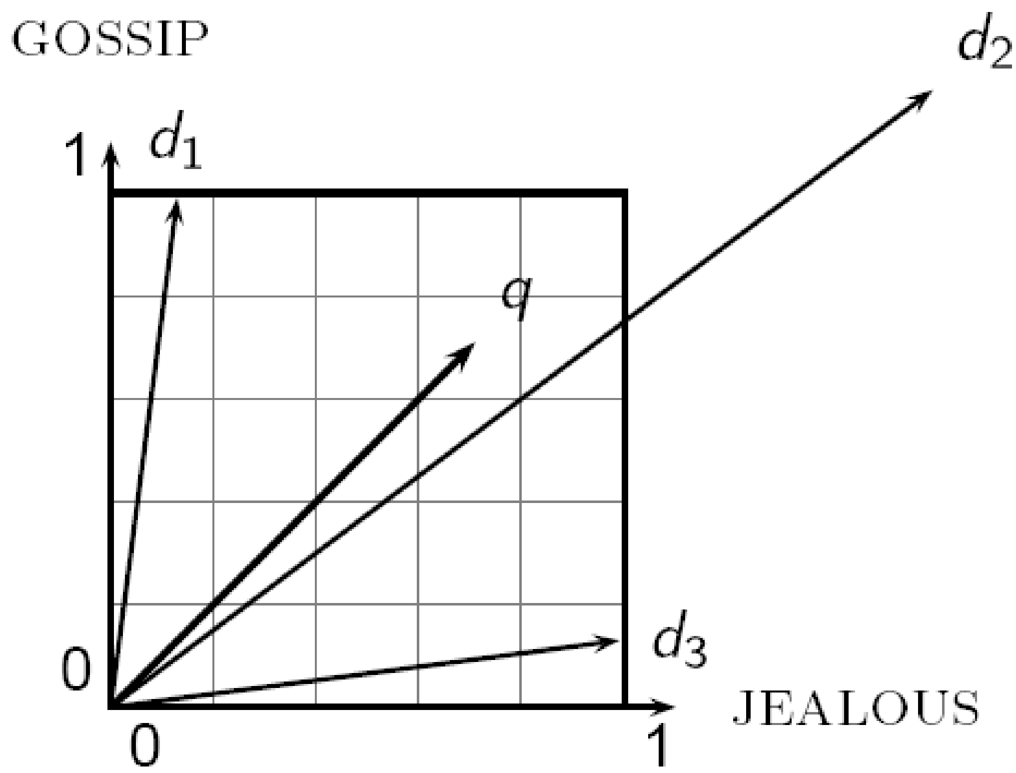
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance

Formalizing vector space proximity

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Use angle instead of distance

- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.
- Key idea: Rank documents according to angle with query.

cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the weight of term *i* in the query
d_i is the weight of term *i* in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos(\vec{q},\vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for q, d length-normalized.

Cosine similarity amongst 3 documents

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

Computing cosine scores

CosineScore(q)

float $Scores[N] = 0$
float $Length[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
    for each pair(d, $tf_{t,d}$) in postings list
    do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array $Length$
for each d
do $Scores[d] = Scores[d] / Length[d]$
return Top K components of $Scores[]$

Computing cosine scores

- Previous algorithm scores term-at-a-time (TAAT)
- Algorithm can be adapted to scoring document-at-a-time (DAAT)
- Storing $w_{t,d}$ in each posting could be expensive
  - …because we'd have to store a floating point number
  - For tf-idf scoring, it suffices to store $tf_{t,d}$ in the posting and idf_t in the head of the postings list
- Extracting the top K items can be done with a priority queue (e.g., a heap)

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log\frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0,\log\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^\alpha$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

BM25

OpenSource Connections

What We Do Case Studies About Us

BM25 The Next Generation of Lucene Relevance

Doug Turnbull — October 16, 2015

There's something new cooking in how Lucene scores text. Instead of the traditional "TF*IDF," Lucene just switched to something called BM25 in trunk. That means a new scoring formula for Solr (Solr 6) and Elasticsearch down the line.

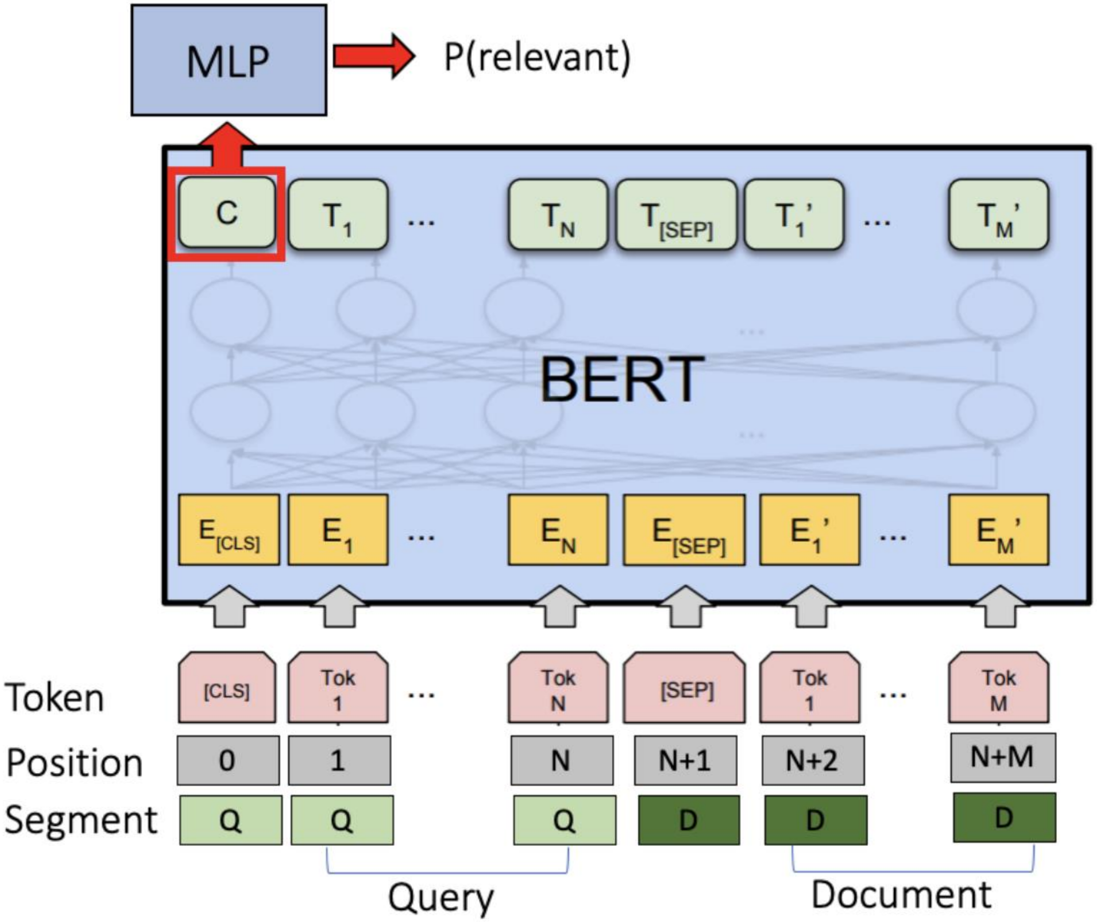
Sounds cool, but what does it all mean? In this article I want to give you an overview of how the switch might be a boon to your Solr and Elasticsearch applications. What was the original TF*IDF? How did it work? What does the new BM25 do better? How do you tune it? Is BM25 right for everything?

Okapi BM25

- BM25 "Best Match 25" (they had a bunch of tries!)
  - Developed in the context of the Okapi system
  - Started to be increasingly adopted by other teams during the TREC competitions
  - It works well

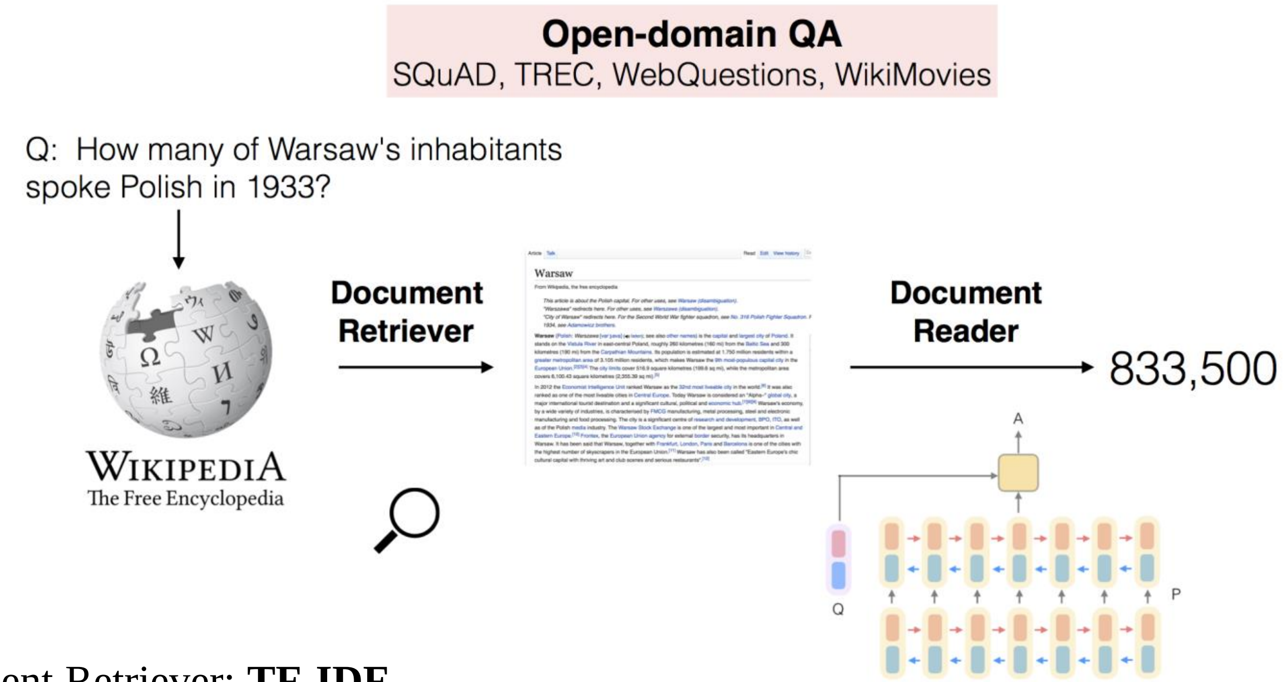
- Goal: be sensitive to term frequency and document length while not adding too many parameters
  - (Robertson and Zaragoza 2009; Spärck Jones et al. 2000)

IR with Contextual Neural Language Modeling



Dai, Zhuyun, and Jamie Callan. "***Deeper text understanding for IR with contextual neural language modeling***" *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2019.

Hệ thống hỏi đáp tự động



Document Retriever: **TF-IDF**

Chen, Danqi, et al. "Reading Wikipedia to Answer Open-Domain Questions." *ACL*. 2017.

Summary – vector space ranking

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Other modern IR models: BM25, BERT, …
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top K (e.g., $K = 10$) to the user

Q&A

Thank you!