DD2476: Search Engines and Information Retrieval Systems

Johan Boye*
KTH

Lecture 4

* Many slides inspired by Manning, Raghavan and Schütze

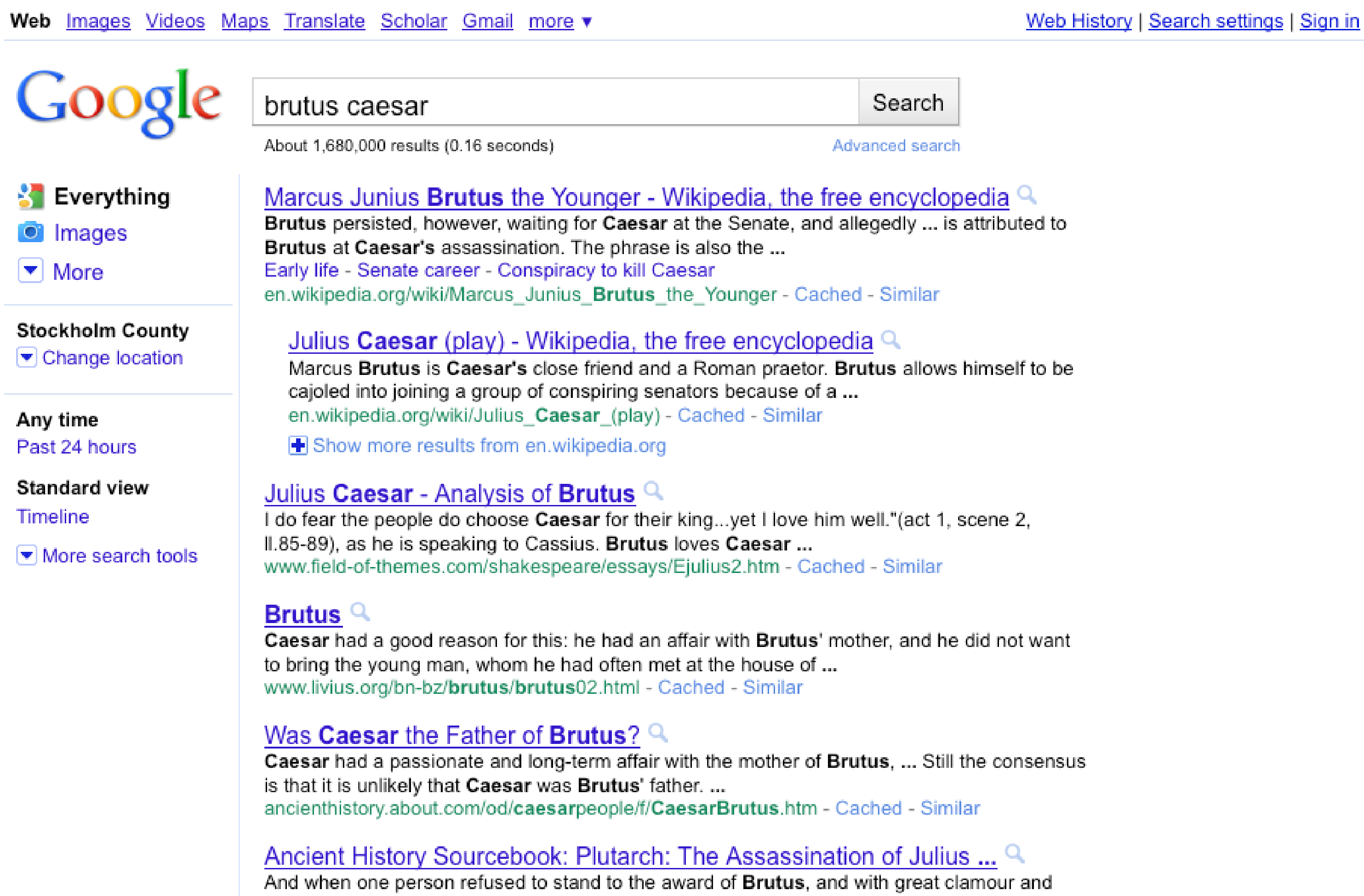
Remember: Boolean retrieval

- In computer assignment 1, you implemented a special case of Boolean retrieval (intersection).
- Boolean retrieval might be good for expert users
- But it is bad for most users, especially for web search.

Problems with Boolean search

- Boolean queries often return **too many** or **too few** results
  - ”zyxel P-660h” → 192 000 results
  - ”zyxel P-660h” ”no card found” → 0 results
- Takes skill to formulate a search query that gives a manageable number of hits.
  - ”AND” gives too few, ”OR” too many
- No ranking of search results

Ranked retrieval



Ranked retrieval

- Every matching document is given a score, say in [0..1]
- The **higher** the score, the **better** the match
- Large result sets do not pose problems
  - Show top *k* results (k≈10)
  - Option to see more.
  - **Premise**: The ranking algorithm works!

Today’s topics

- The **vector space model**
  - documents and queries are represented as vectors in a high-dimensional space
- **tf_idf weighting**
  - take the frequency and informativeness of terms into account

Term-document incidence matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	0	1	1	1
citizen	1	1	0	0	1	0

1 if term is present in document, 0 otherwise

Word count matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
citizen	1	2	0	0	1	0

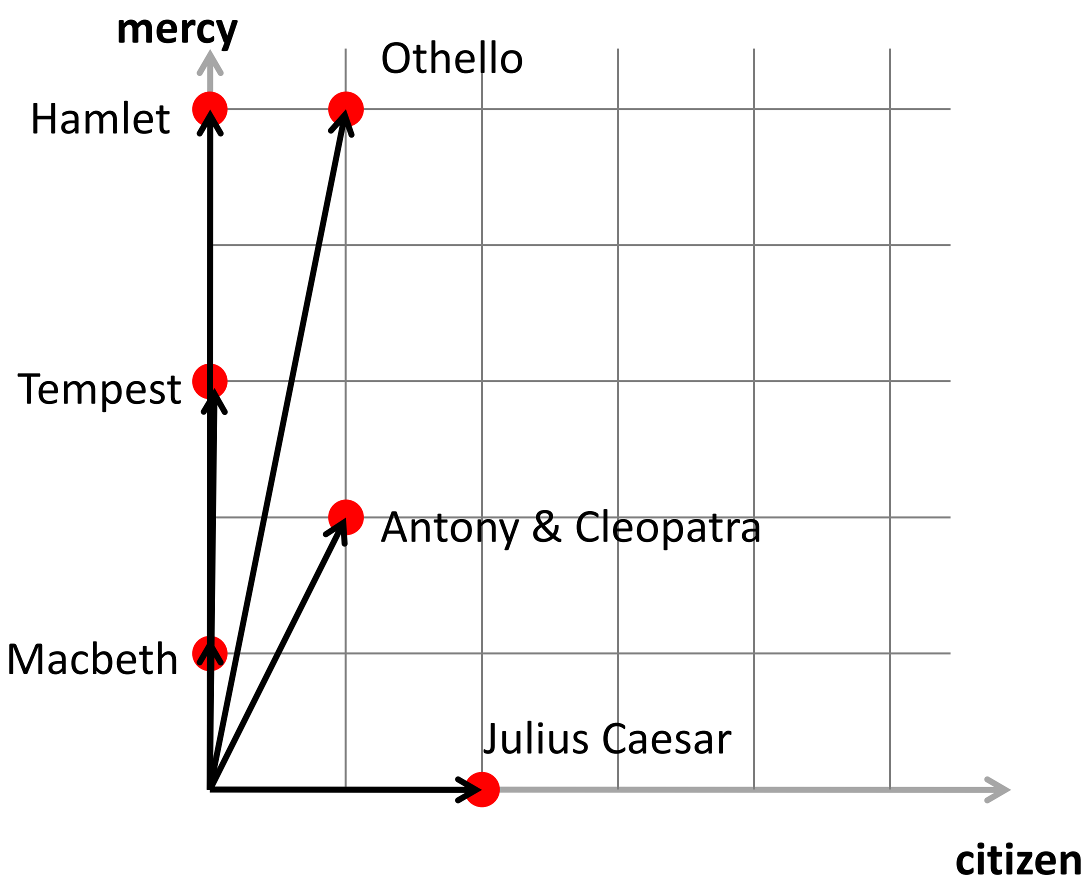
Every document is a vector in term space.

Word count matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
citizen	1	2	0	0	1	0

Let's have a look at these dimensions only.

Documents as vectors



Bag-of-words

- Represent documents as vectors

$$d = (c_1, c_2, \ldots, c_n)$$

where c_i is the number of occurrences of word w_i

- Called a **bag-of-words** representation ('bag' = multiset)
- Ordering of words **not** considered
  - "Carl is wiser than Mary" and "Mary is wiser than Carl" have the **same** vector

Documents as vectors

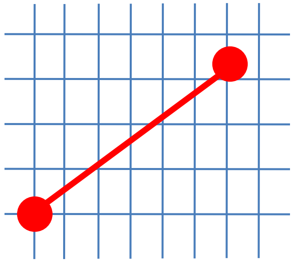
- So we have a |V|-dimensional space
  - Terms are axes/dimensions
  - Documents are points/vectors in this space
- Very high-dimensional
  - ~195,000 dimensions for our davisWiki corpus, much more for entire web
- Very sparse vectors - most entries zero
- How can we compare such vectors?

Cosine similarity

Comparing points (vectors)

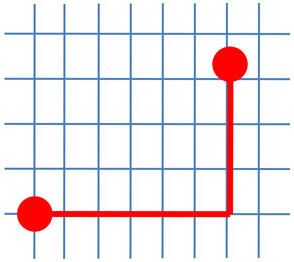
- **Euclidean** distance between $u = (u_1 \ldots u_n)$ and $v=(v_1 \ldots v_n)$

$$\sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}$$

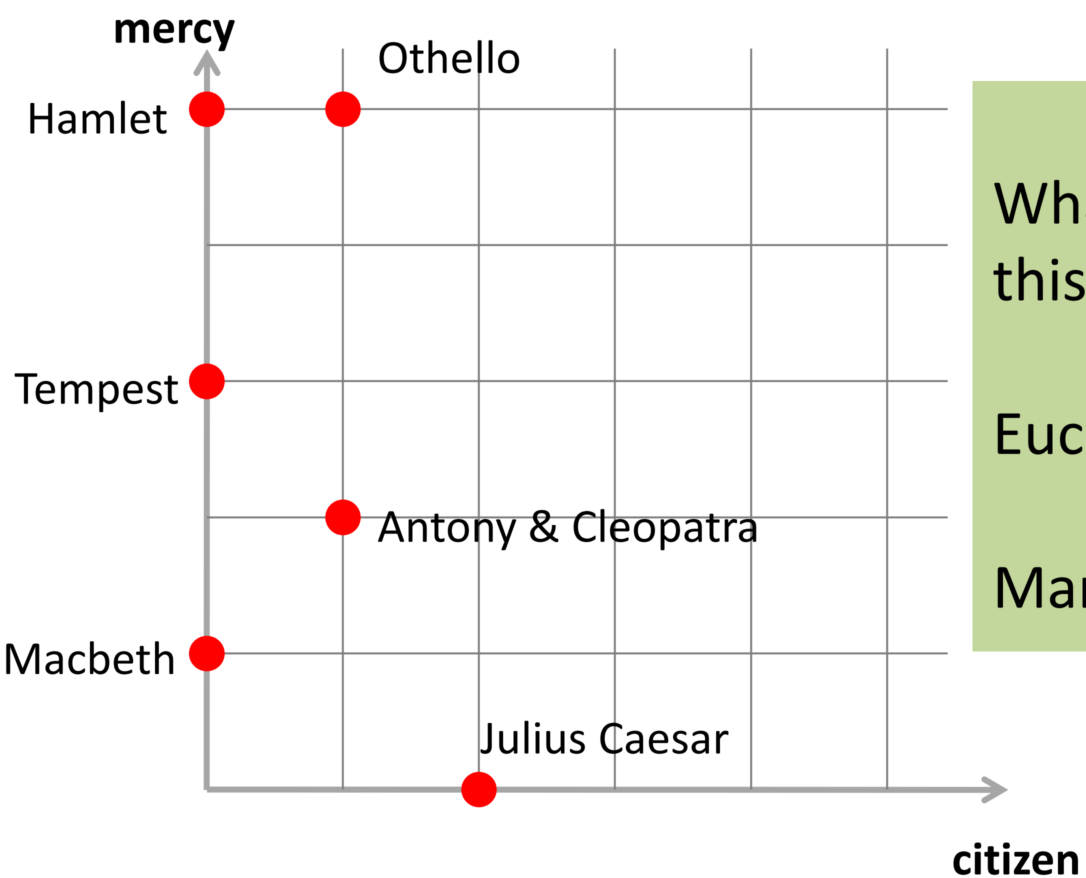


- **Manhattan** distance between u and v:

$$\sum_{i=1}^{n} \left| u_i - v_i \right|$$



Documents as vectors

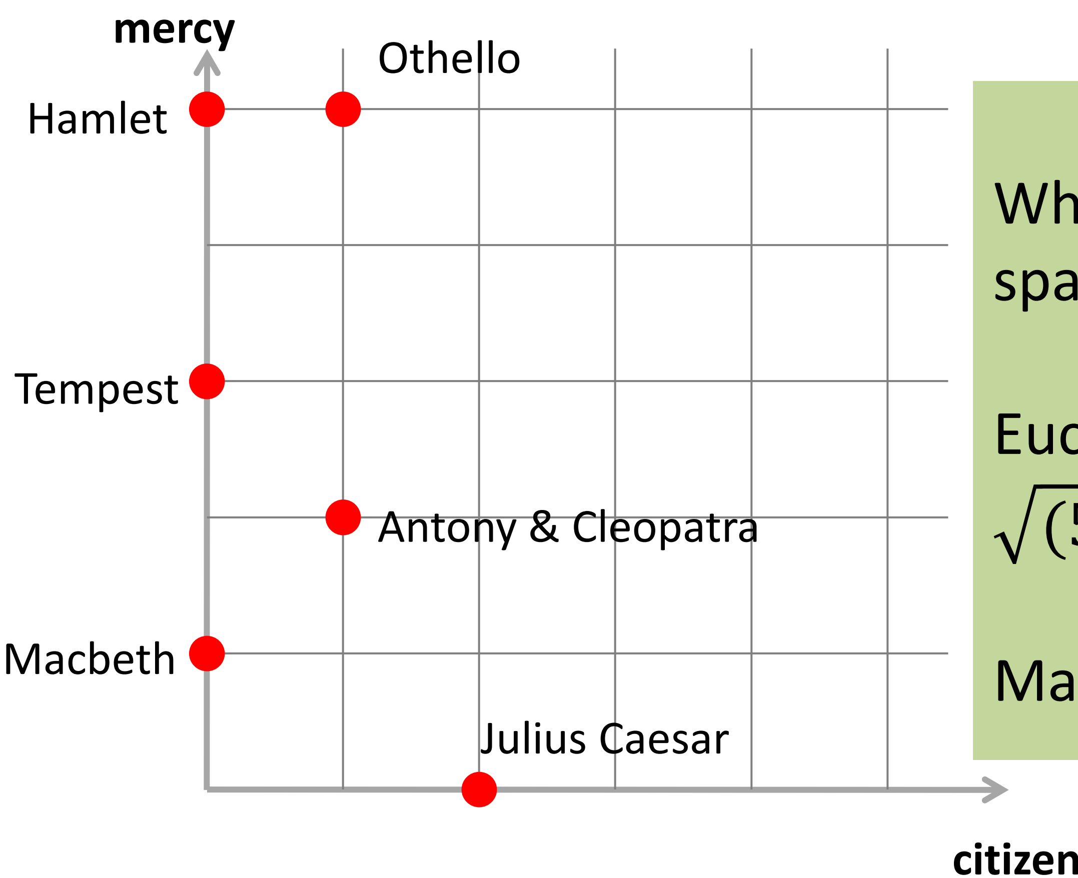


What is d(Hamlet, A&C) in this space?

Euclidean:

Manhattan:

Documents as vectors



What is d(Hamlet, A&C) in this space?

Euclidean:
$\sqrt{(5-2)^2+(0-1)^2} = \sqrt{10}$

Manhattan: |5-2| + |0-1| = 4

What’s the point?

- Suppose we have the query

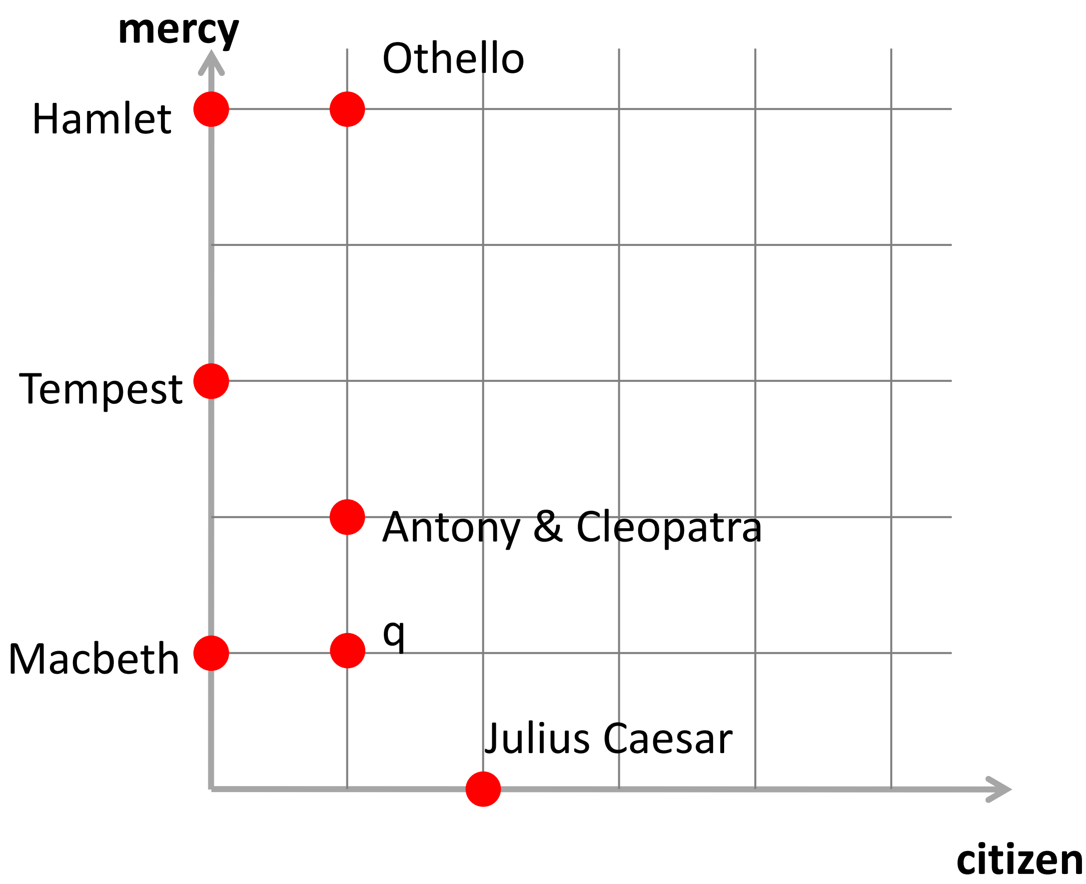
```
mercy citizen
```

- This query can be represented as the vector q=(1,1) (in the mercy-citizen space).
- Perhaps the most relevant documents are those closest to the query?

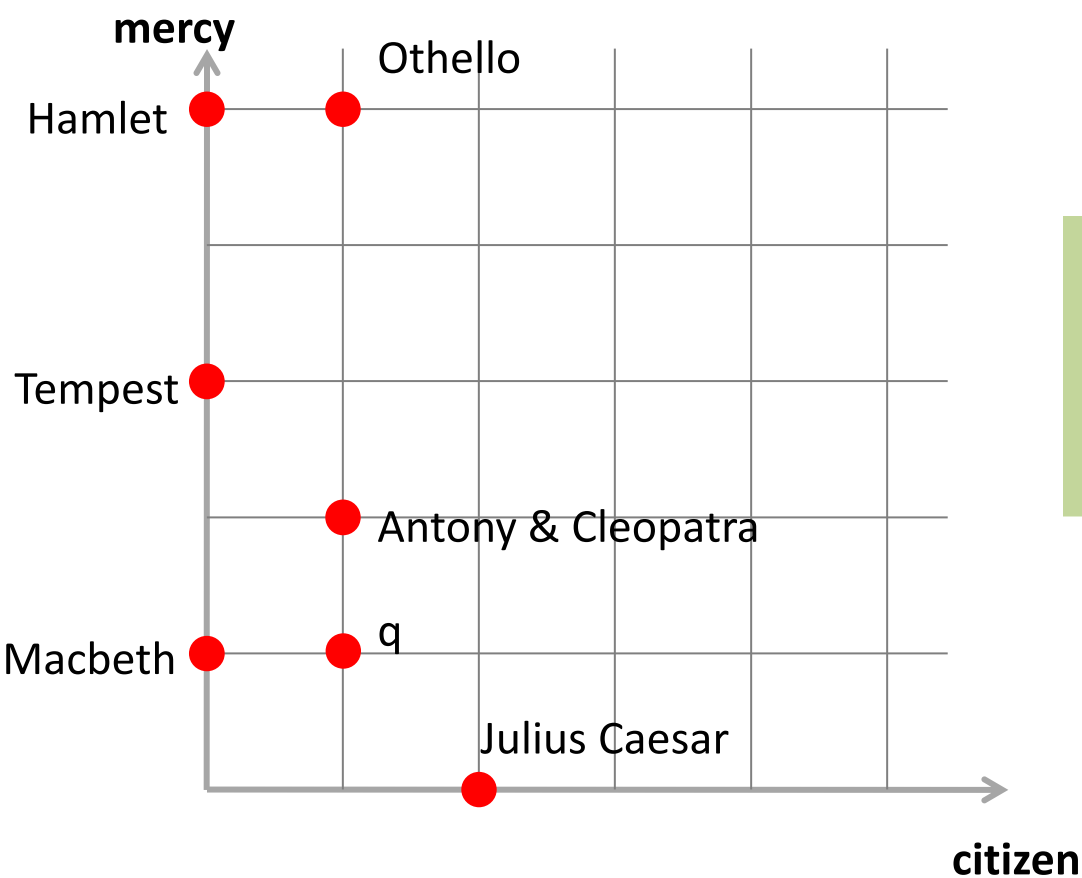
Queries as vectors

- **Key idea 1:** Represent queries as vectors in same space
- **Key idea 2:** Rank documents according to proximity to query in this space
- Recall:
  - Get away from Boolean model
  - Rank more relevant documents higher than less relevant documents

Documents as vectors



Documents as vectors



A&C and Macbeth are closest to q.

Short distance = high relevance?

- However consider the query

```
information retrieval
```

- and the 2 documents:
  - the Wikipedia article on Information Retrieval
  - “blue fish”
- Which is most relevant?
- Which is closest to the query?

Dot product

- Recall: the **dot product** of two vectors is

$$u \cdot v = \sum_{i=0}^{n} u_i v_i$$

- e.g. the dot product of “information retrieval” and “blue fish” will be 0.

”information retrieval”	(0,0,…,1,…0,…,0,…,1,…)
”blue fish”	(0,0,…,0,…1,…,1,…,0,…)

Dot product

$$u \cdot v = \sum_{i=0}^{n} u_i v_i$$

- The dot product of “information retrieval” and the Wikipedia article on IR will be large (=196)

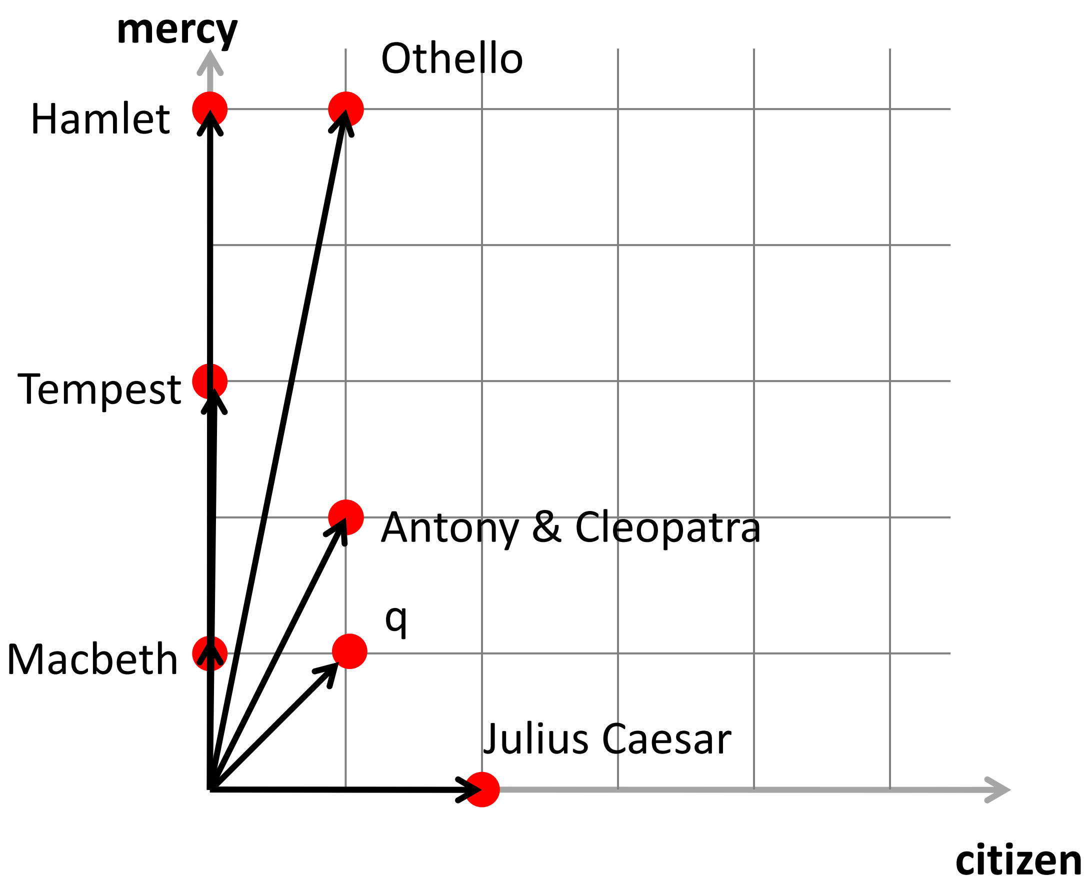
”information retrieval”	(0,0,...,1,............1,...)
the Wiki IR article	(0,0,..,103,.........93..)

Large dot product = high relevance?

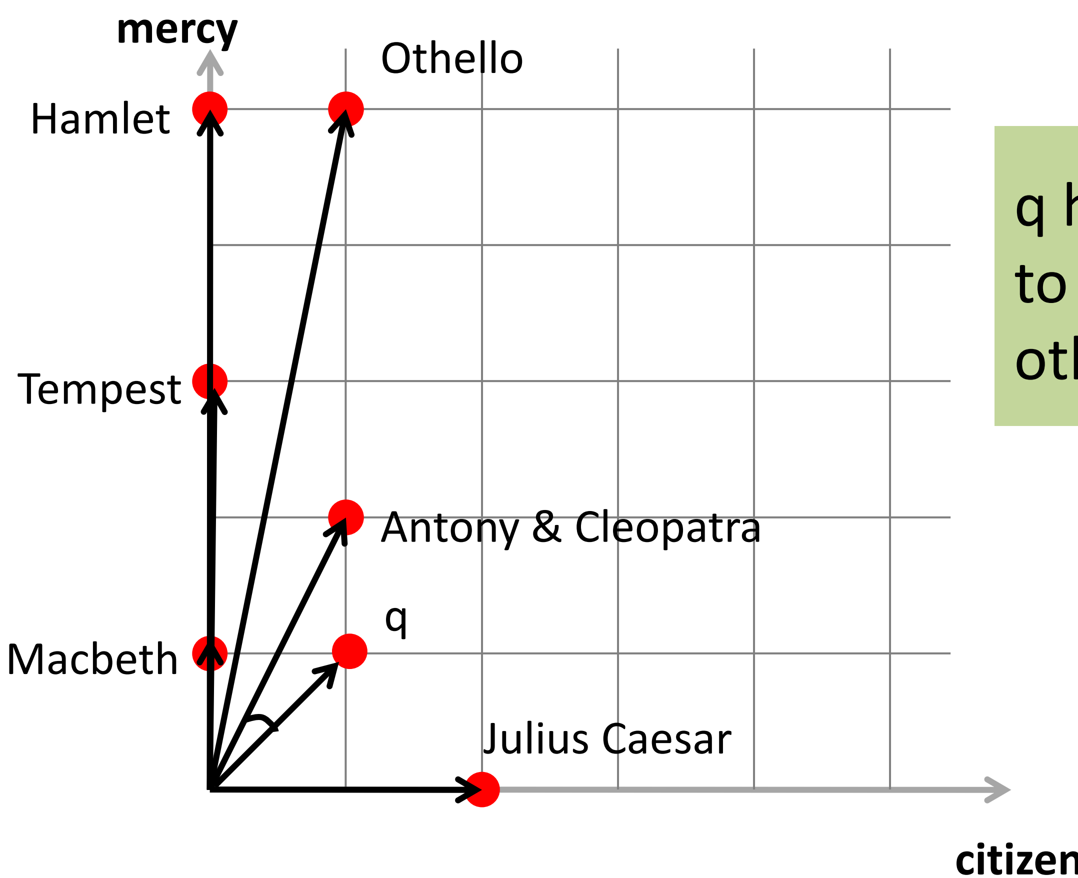
- So should we use the dot product as a rating mechanism?

- However, only using the dot product will favour long documents (why)?

Documents as vectors



Documents as vectors



q has a smaller angle to A&C is than to any other document.

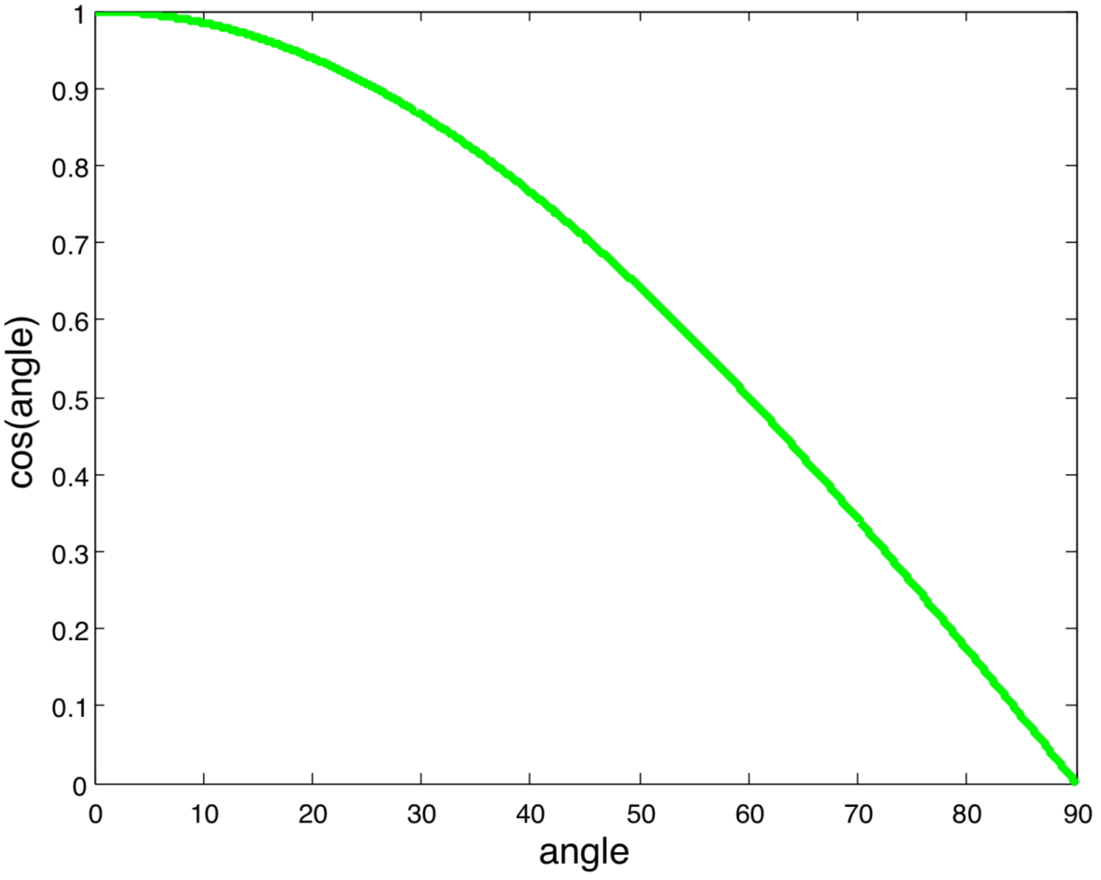
Small angle= high relevance?

- Small angle between two docs d1 and d2 = the **distribution of terms** is **similar** in d1 and d2

- So should we use the angle as a rating mechanism?
  - Small angle with q = higher rating?

- In fact, we will use the **cosine** of the angle (rather than the angle itself)

Graph of cos(angle)



Monotonically decreasing

Cosine similarity

Dot product of *u* and *v*:

$$u \cdot v = \sum_{i=0}^{n} u_i v_i$$

It holds that:

$$u \cdot v = \|u\| \|v\| \cos\theta$$

where |*u*| = the length of *u*, and θ = angle between *u* and *v*

Therefore:

$$\cos\theta = \frac{\sum_{i=0}^{n} u_i v_i}{\|u\| \|v\|}$$

Length of a vector

- There is more than one length norm:
  - Manhattan

$$\|x\|_1 = \sum_i |x_i|$$

  - Euclidean

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

Cosine similarity

Dot product   Unit vectors

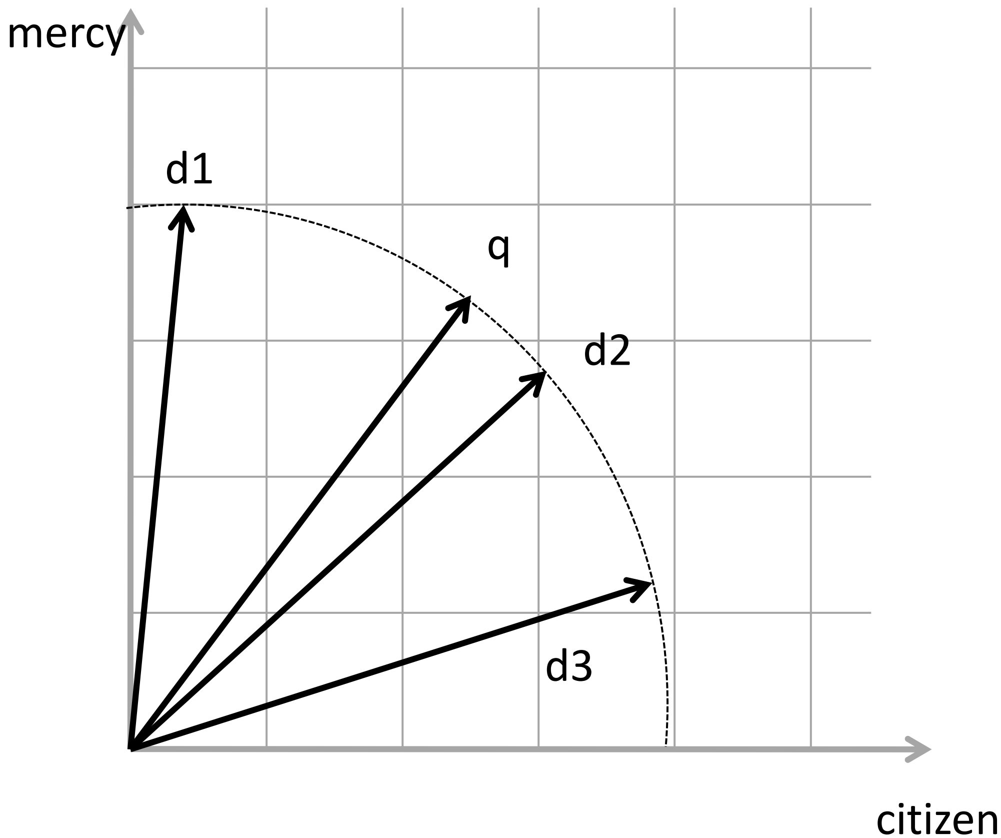
$$\cos(q,d) = \frac{q \cdot d}{|q||d|} = \frac{q}{|q|} \cdot \frac{d}{|d|} = \frac{\sum_{i=0}^{n} q_i d_i}{\sqrt{\sum_{i=0}^{n} q_i^2}\sqrt{\sum_{i=0}^{n} d_i^2}}$$

cos(q,d) = is the **cosine similarity** of *q* and *d*

= cosine of the angle between *q* and *d*

= dot product of the unit vectors *q*/|*q*| and *d*/|*d*|

Vectors after length normalisation



Quiz: Cosine similarity

d1	to be or not to be
d2	to be is to do
d3	i do i do i do i do i do
d4	do be do be do

- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?
- What is cos(d3, d4)?

Quiz: Cosine similarity

d1	to be or not to be	(2,2,1,1,0,0,0)
d2	to be is to do	(2,1,0,0,1,1,0)
d3	i do i do i do i do i do	(0,0,0,0,0,5,5)
d4	do be do be do	(0,2,0,0,0,3,0)

- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?
- What is cos(d3, d4)?

Quiz: Cosine similarity

d1	to be or not to be	(2,2,1,1,0,0,0)
d2	to be is to do	(2,1,0,0,1,1,0)
d3	i do i do i do i do i do	(0,0,0,0,0,5,5)
d4	do be do be do	(0,2,0,0,0,3,0)

- What do the d1-d4 vectors look like?
- What is cos(d1, d2)? $\frac{2 \cdot 2 + 2 \cdot 1}{\sqrt{2^2+2^2+1^2+1^2}\sqrt{2^2+1^2+1^2+1^2}} = \frac{6}{\sqrt{70}}$
- What is cos(d3, d4)?

Quiz: Cosine similarity

d1	to be or not to be	(2,2,1,1,0,0,0)
d2	to be is to do	(2,1,0,0,1,1,0)
d3	i do i do i do i do i do	(0,0,0,0,0,5,5)
d4	do be do be do	(0,2,0,0,0,3,0)

- What do the d1-d4 vectors look like?
- What is cos(d1, d2)? $\frac{2\cdot 2+2\cdot 1}{\sqrt{2^2+2^2+1^2+1^2}\sqrt{2^2+1^2+1^2+1^2}} = \frac{6}{\sqrt{70}}$
- What is cos(d3, d4)?

$$\frac{5\cdot 3}{\sqrt{5^2+5^2}\sqrt{2^2+3^2}} = \frac{3}{\sqrt{26}}$$

The tf_idf weighting scheme

Word count matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
citizen	1	2	0	0	1	0

Term frequencies **tf**

log-frequency weighting

- Which numbers should fill our vectors?
- Raw term frequency might not be what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term
  - But perhaps not 10 times more relevant
- **Log-frequency weight** of term *t* in document *d*

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Example

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

term	$\mathbf{tf_{t,d}}$	$\mathbf{w_{t,d}}$
airplane	0	
shakespeare	1	
calpurnia	10	
under	100	
the	1,000	

Document frequency ***df***

- Rare terms are more informative than frequent terms
- Example: rare word CAPRICIOUS
  - Document containing this term is very likely to be relevant to query CAPRICIOUS
    → High weight for rare terms like CAPRICIOUS
- Example: common word THE
  - Document containing this term can be about anything
    → Very low weight for common terms like THE
- We will use **document frequency** (df) to capture this.

idf (inverse ***df***)

- Informativeness ***idf*** (inverse document frequency) of t:

$$\mathrm{idf}_t = \log(N/\mathrm{df}_t)$$

where N is the number of documents.

log (N/df_t) instead of N/df_t to “dampen” the effect of idf.

tf_idf weighting: Example

Query: THE CAPRICIOUS PERSON

Doc contains 10 ‘the’, 1 ‘capricious, 1 ‘person’

idf(the) = 0.22, idf(capricious) = 8.52, idf(person)=6.21

Dot product without idf:

$10+1+1 = 12$

With idf:

$10 \times 0.22 + 1 \times 8.52 + 1 \times 6.21 = 14.95$

tf_idf weighting

- **tf_idf weight** of a term: product of tf weight and idf weight

- Best known weighting scheme in information retrieval

- Increases with the **number of occurrences** within a document
- Increases with the **rarity of the term** in the collection

Effect of idf on ranking

- Note that idf has no effect on ranking for one-term queries, like 'CAPRICIOUS'.

- Only effect for >1 term
  - Query THE CAPRICIOUS PERSON: idf puts more weight on CAPRICIOUS than PERSON…
  - … and much more than THE

Cosine similarity again

Dot product   Unit vectors

$$\cos(q,d) = \frac{q \cdot d}{|q||d|} = \frac{q}{|q|} \cdot \frac{d}{|d|} = \frac{\sum_{i=0}^{n} q_i d_i}{\sqrt{\sum_{i=0}^{n} q_i^2}\sqrt{\sum_{i=0}^{n} d_i^2}}$$

q_i is the tf-idf weight of term *i* in the query

d_i is the tf-idf weight of term *i* in the document

tf_idf-weighting - Example

d1	to be or not to be
d2	to be is to do
d3	i do i do i do i do i do
d4	do be do be do

- What is idf(to)?
- What is tf(d1,to)?
- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?

tf_idf-weighting - Example

d1	to be or not to be
d2	to be is to do
d3	i do i do i do i do i do
d4	do be do be do

- What is idf(to)? log(4/2) = 0.7
- What is tf(d1,to)?
- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?

tf_idf-weighting - Example

d1	to be or not to be
d2	to be is to do
d3	i do i do i do i do i do
d4	do be do be do

- What is idf(to)? log(4/2) = 0.7
- What is tf(d1,to)? 2
- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?

tf_idf-weighting - Example

d1	to be or not to be	(1.4, 0.6, 1.4, 1.4, 0, 0, 0)
d2	to be is to do	(1.4, 0.3, 0, 0, 1.4, 0.3, 0)
d3	i do i do i do i do i do	(0, 0, 0, 0, 0, 1.5, 6.9)
d4	do be do be do	(0, 0.6, 0, 0, 0, 0.9, 0)

- What is idf(to)? log(4/2) = 0.7
- What is tf(d1,to)? 2
- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?

tf_idf-weighting - Example

d1	to be or not to be	(1.4, 0.6, 1.4, 1.4, 0, 0, 0)
d2	to be is to do	(1.4, 0.3, 0, 0, 1.4, 0.3, 0)
d3	i do i do i do i do i do	(0, 0, 0, 0, 0, 1.5, 6.9)
d4	do be do be do	(0, 0.6, 0, 0, 0, 0.9, 0)

- What is idf(to)? log(4/2) = 0.7
- What is tf(d1,to)? 2
- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?

$$\frac{1.4 \times .1.4 + 0.6 \times 0.3}{\sqrt{1.4^2 + 0.6^2 + 1.4^2 + 1.4^2}\sqrt{1.4^2 + 0.3^2 + .1.4\ ^2 + 0.3^2}} \approx 0.42$$

Summary – Vector space model

- Vector space model:
  - Represent the query as a tf-idf vector
  - Represent each document as a tf-idf vector
  - 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

Computing cosine scores

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} × 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

- In the code skeleton for the assignments...
- … in the **`Index.java`** interface…
- … there is a HashMap **`docLengths`** that stores the number of tokens (=Manhattan length) for all documents.
- This is computed for you at indexing time.
- In task 2.6, you will need to compute Euclidean lengths for all documents.

Weighting schemes

- Different weighting schemes:

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)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

- In assignment 2.3 you will explore some of these variants

Computing cosine scores efficiently

- Approximation:
  - Assume that terms only occur once in the query

$$w_{t,q} \leftarrow \begin{cases} 1, & \text{if } \mathrm{tf}_{t,q} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- Works for short queries (|q| << *N*)
- Works since ranking is only relative
  - We only care about the order, not the actual numbers

Computing cosine scores efficiently

FASTCOSINESCORE(q)

float $Scores[N] = 0$
for each d
do Initialize $Length[d]$ to the length of doc d
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 add $wf_{t,d}$ to $Scores[d]$
Read the array $Length[d]$
for each d
do Divide $Scores[d]$ by $Length[d]$
return Top K components of $Scores[]$

Figure 7.1 A faster algorithm for vector space scores.

Computing cosine scores efficiently

- Downside of approximation: **sometimes get it wrong**
  - A document not in the top *K* may creep into the list of *K* output documents
- Is this such a bad thing?
- Cosine similarity is only a proxy
  - User has a task and a query formulation
  - Cosine matches documents to query
  - Thus cosine is anyway a proxy for user happiness
  - If we get a list of *K* documents "close" to the top *K* by cosine measure, should be ok

Choosing K largest scores efficiently

- Do we really need to order *every* document?
- No, usually it is enough to retrieve top *K* documents wrt query

- Do selection:
  - avoid visiting all documents

- There are several schemes that achieves this

Index elimination, version 1

- Disregard low-idf terms
- Example:

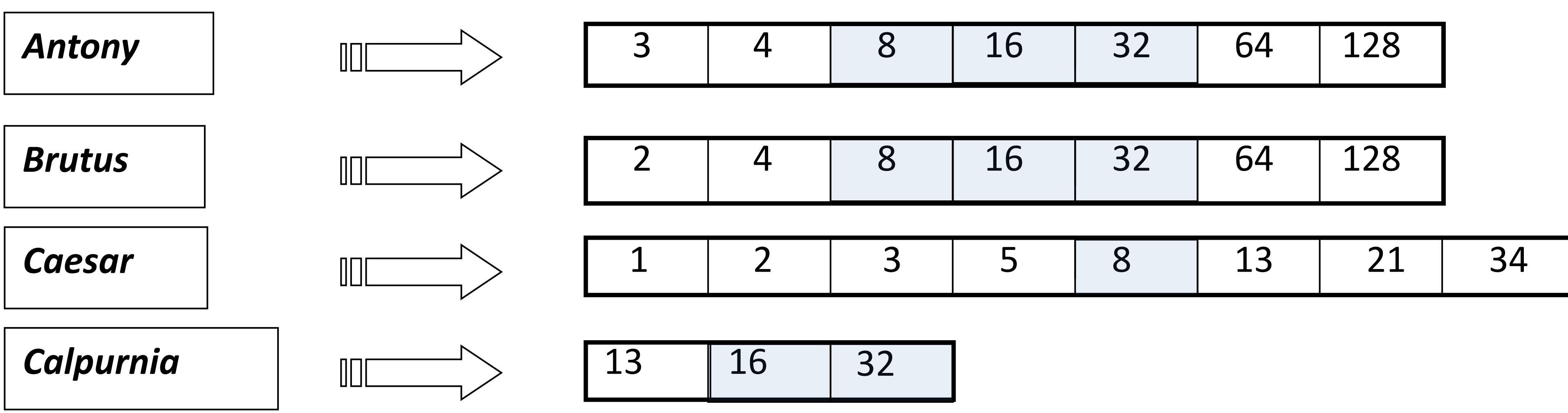
  CATCHER IN THE RYE
- **Only accumulate scores from CATCHER and RYE**
- Intuition:
  - IN and THE contribute little to the scores – do not alter rank-ordering much
- Benefit:
  - We will consider far fewer documents
  - (since postings lists of low-idf terms have many documents)

Index elimination, version 2

- Example:

  CAESAR ANTONY CALPURNIA BRUTUS

- **Only compute scores for documents containing ≥3 query terms**



Champions lists

- Precompute for each dictionary term *t,* the *r* documents of highest tf-idf$_{td}$ weight
  - Call this the **champions list** (fancy list, top docs) for *t*

- Benefit:
  - At query time, only compute scores for documents in the champion lists – fast

- Issue:
  - *r* chosen at index build time
  - Too large: slow
  - Too small: too few results

Query parser

- Query phrase:

  RISING INTEREST RATES

- Sequence:
  - Run as a **phrase query**
  - If <*K* documents contain the phrase RISING INTEREST RATES, run **phrase queries** RISING INTEREST and INTEREST RATES
  - If still <K docs, run **vector space query** RISING INTEREST RATES
  - Rank matching docs by **vector space scoring**

Static quality scores

- We want top-ranking documents to be both **relevant** and **authoritative**
  - Relevance – cosine scores
  - Authority – query-independent property
- Assign **query-independent quality score** g(*d*) in [0,1] to each document *d*
- net-score(q,d) = w_1*g(d) + w_2*cos(q,d)
  - Two "signals" of user happiness

BM25 ranking

$$bm25(d) = \sum_{t \in q} \log\left(\frac{N}{df_t}\right) \frac{(k+1)tf_{td}}{k(1 - b + b\frac{L_d}{L_{ave}}) + tf_{td}}$$

d is a document, *q* is the query

L_d is the length of *d*, L_{ave} is the average length of a doc

The *b* parameter controls length scaling

The *k* parameter controls frequency scaling

BM25 ranking

$$bm25(d) = \sum_{t \in q} \log\left(\frac{N}{df_t}\right) \frac{2tf_{td}}{2\frac{L_d}{L_{ave}} + tf_{td}}$$

Special case: b=1 and k=1

Shorter documents will be up-ranked, longer docs will be down-ranked