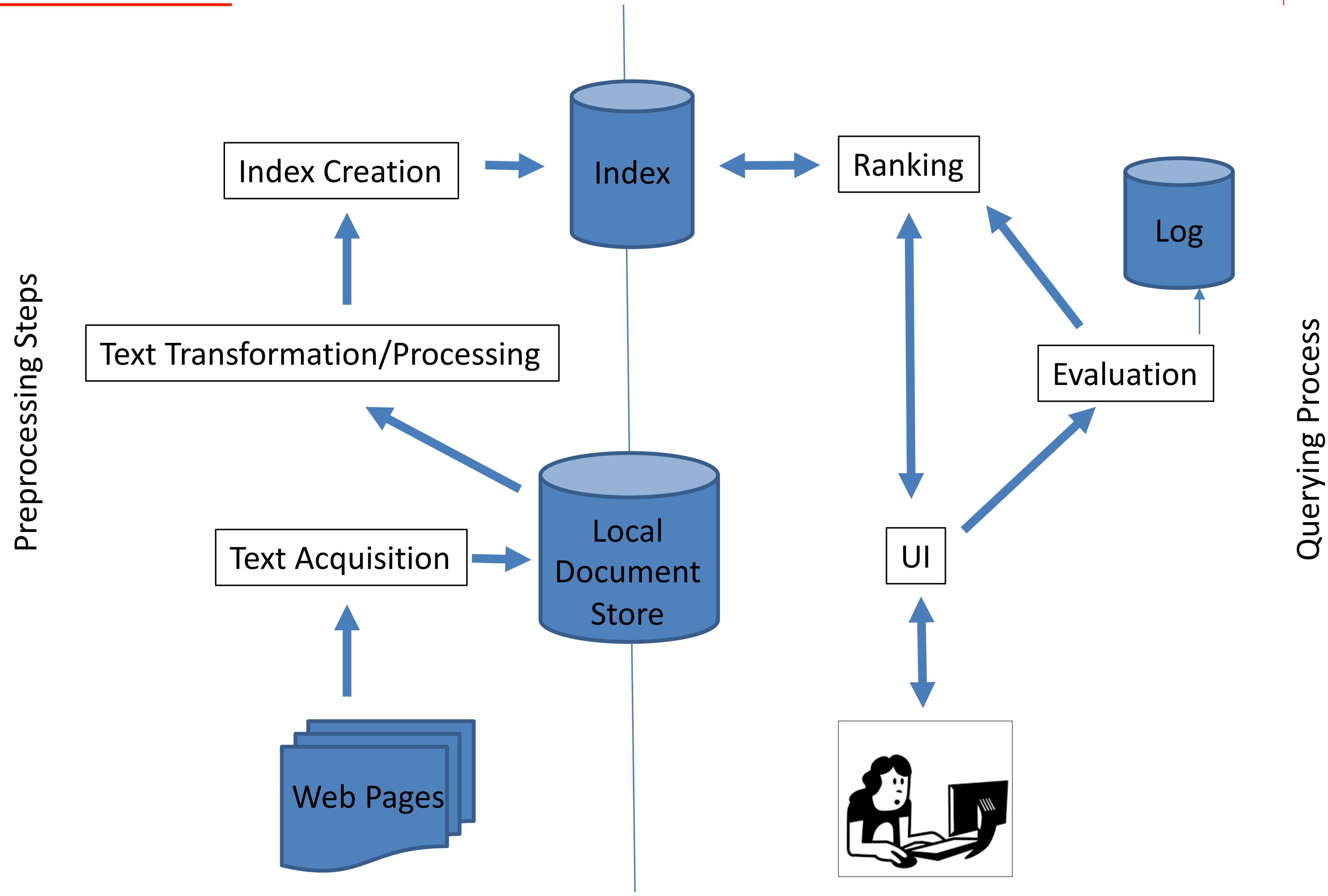
Informatics 225
Computer Science 221

Information Retrieval

Lecture 21

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. Powerpoint theme by Prof. André van der Hoek.

Architecture



Vector space model

Binary → count → weight matrix

Term-document incidence → Term-document count matrix → term-document weight matrix

Binary → count → weight matrix

Term-document incidence → Term-document count matrix → term-document weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Documents as vectors

- So we have a |V|-dimensional vector space

Documents as vectors

- So we have a |V|-dimensional vector space
- Terms are axes of the space

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

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*

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

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

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

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

- Recall:
  - We want to get away from the you're-either-in-or-out Boolean model.
  - Instead: rank more relevant documents higher than less relevant documents

Formalizing vector space proximity

- First thing that comes to mind: distance between two points
  - (= distance between the end points of the two vectors)

Formalizing vector space proximity

- First thing that comes to mind: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?

Formalizing vector space proximity

- First thing that comes to mind: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?
  - Euclidean distance is a bad idea . . .

Formalizing vector space proximity

- First thing that comes to mind: distance between two points
  – (= distance between the end points of the two vectors)

- Euclidean distance?

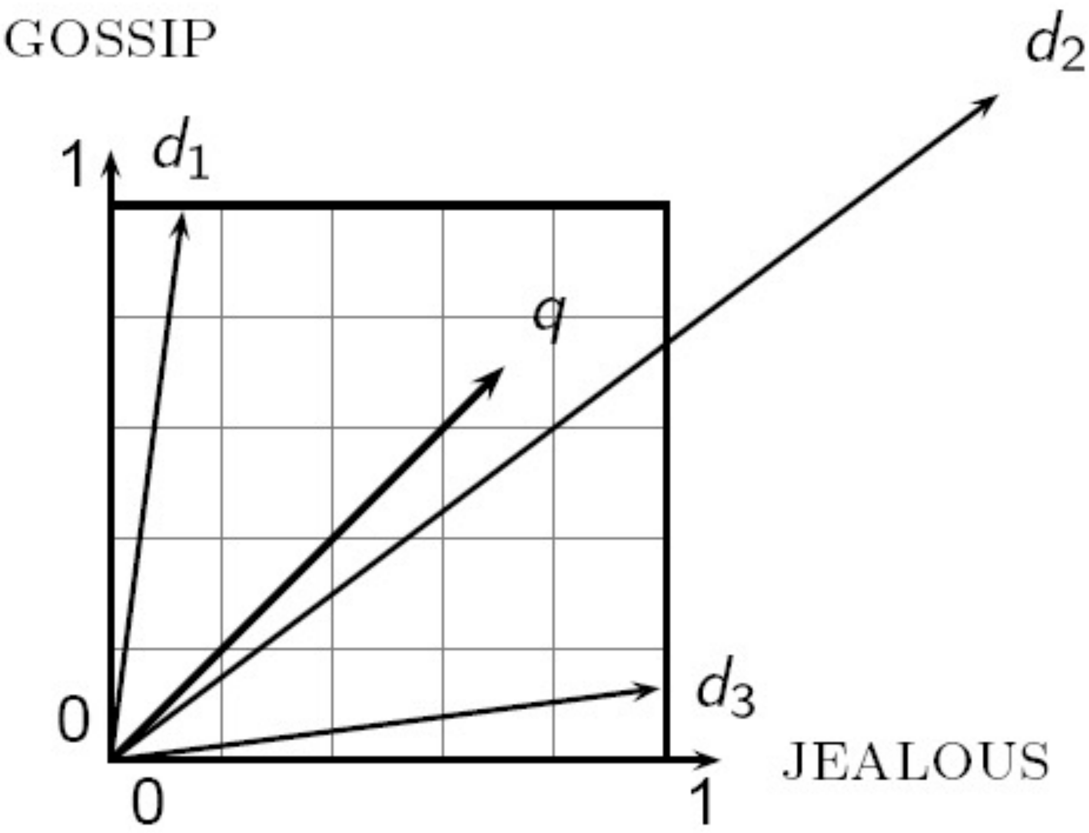
  – Euclidean distance is a bad idea . . .

  . . . because Euclidean distance is large for vectors of different lengths.

  and queries and documents will (usually) have very different lengths…

Why Euclidean distance is (often) 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.



What can be better?

- Thought experiment: take a document *d* and append it to itself. Call this document *d*’.

What can be better?

- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
  - “Semantically” d and d’ have the same content

What can be better?

- Thought experiment: take a document *d* and append it to itself. Call this document *d*'.
  - "Semantically" d and d' have the same content
  - The Euclidean distance between the two documents can be quite large : *bad!*

Use angle instead of distance

- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
  - “Semantically” d and d′ have the same content
  - The Euclidean distance between the two documents can be quite large : *bad!*
  - The angle between the two documents is 0, corresponding to maximal similarity : *good!*

Use angle instead of distance

- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
  - “Semantically” d and d’ have the same content
  - The Euclidean distance between the two documents can be quite large : *bad!*
  - The angle between the two documents is 0, corresponding to maximal similarity : *good!*
- **Key idea:** Rank documents according to the <u>angles between documents and query</u>.

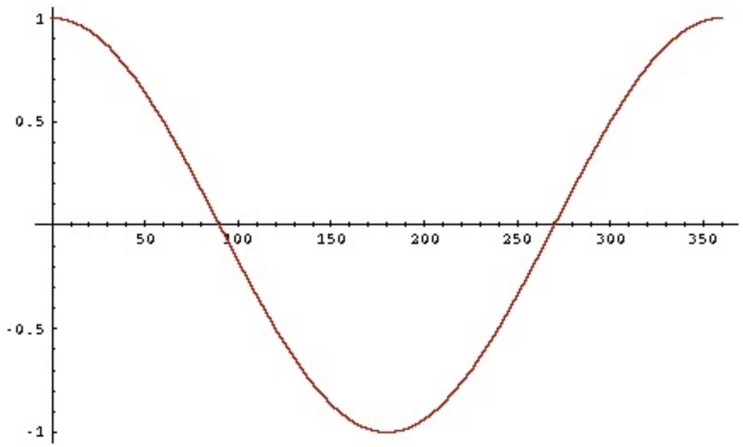
From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of angle(query, document)
  - Rank documents in increasing order of cosine(query, document)

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in <u>decreasing order of angle(query, document)</u>
  - Rank documents in <u>increasing order of cosine(query, document)</u>

- Cosine is a monotonically decreasing function for the interval [0°, 180°]



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 tf-idf weight of term i in the query
d_i is the tf-idf 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 $\vec{d}$.

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

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

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

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

- Dividing a vector by its L_2 norm makes it a unit (length) vector (*on surface of unit hypersphere*)

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

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

- Dividing a vector by its L_2 norm makes it a unit (length) vector (*on surface of unit hypersphere*)

- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

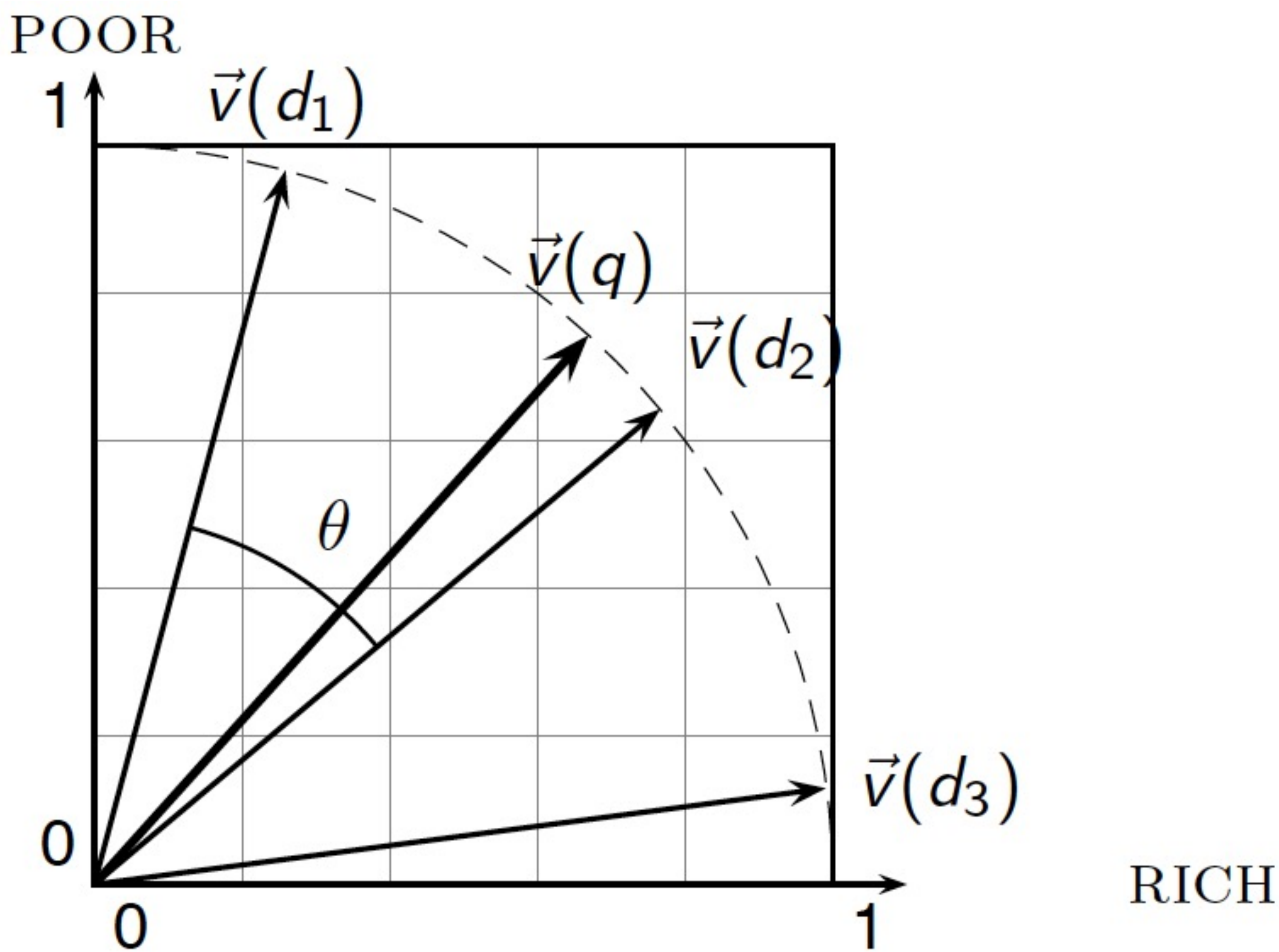
Cosine for length-normalized vectors

- 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 illustrated



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$(d, \mathrm{tf}_{t,d})$ 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 $Scores[]$

Cosine similarity amongst 3 documents

- **How similar are the novels?**
  - SaS: *Sense and Sensibility*
  - PaP: *Pride and Prejudice*
  - 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)

Cosine similarity amongst 3 documents

- How similar are the novels?
  - SaS: *Sense and Sensibility*
  - PaP: *Pride and Prejudice*
  - 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

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	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

3 documents example contd.

Log frequency weighting

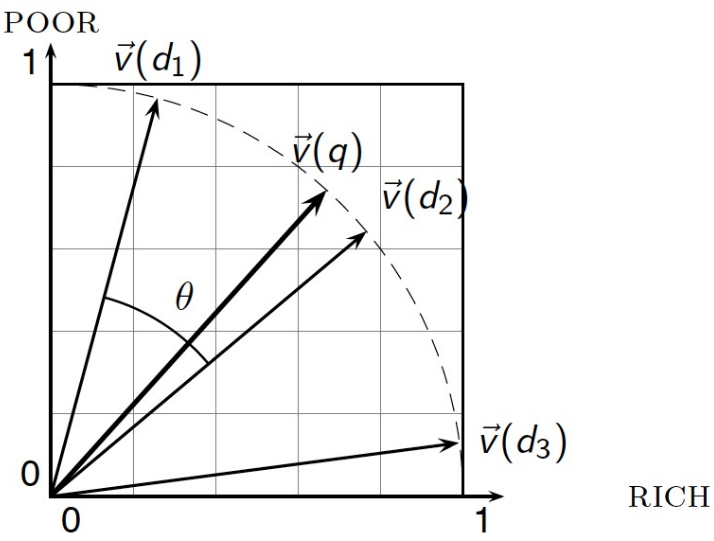
term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

Documents are already in the surface of a hypersphere, so

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



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	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.557 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94

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	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.557 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

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	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.557 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

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)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

ltc : the general standard that is usually adopted

In parenthesis: acronyms for weight schemes.

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table
- **A very standard weighting scheme is: lnc.ltc**

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table
- **A very standard weighting scheme is: lnc.ltc**
- **Document**: logarithmic tf (l as first character), no idf and cosine normalization
- **Query**: logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization …

tf-idf example: lnc.ltc (remember : ddd.qqq)

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0						1				
best	1						0				
car	1						1				
insurance	1						2				

tf-idf example: lnc.ltc (remember : ddd.qqq)

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0					1	1			
best	1	1					0	0			
car	1	1					1	1			
insurance	1	1					2	1.3			

tf-idf example: lnc.ltc (remember : ddd.qqq)

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3			1	1			
best	1	1	50000	1.3			0	0			
car	1	1	10000	2.0			1	1			
insurance	1	1	1000	3.0			2	1.3			

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)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

ltc : standard

In parenthesis: acronyms for weight schemes.

tf-idf example: lnc.ltc (remember : ddd.qqq)

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0		1	1	1		
best	1	1	50000	1.3	1.3		0	0	0		
car	1	1	10000	2.0	2.0		1	1	1		
insurance	1	1	1000	3.0	3.0		2	1.3	1.3		

tf-idf example: lnc.ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	

Doc length $=\sqrt{1^2+0^2+1^2+1.3^2} \approx 1.92$

tf-idf example: lnc.ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Score = 0+0+0.27+0.53 = 0.8

Summary – vector space ranking

1. Represent the query as a weighted tf-idf vector

2. Represent each document as a weighted tf-idf vector

Summary – vector space ranking

1. Represent the query as a weighted tf-idf vector

2. Represent each document as a weighted tf-idf vector

3. Compute the cosine similarity score for the query vector and each document vector

Summary – vector space ranking

1. Represent the query as a weighted tf-idf vector

2. Represent each document as a weighted tf-idf vector

3. Compute the cosine similarity score for the query vector and each document vector

4. Rank documents with respect to the query by score

5. Return the top K (e.g., $K = 10$) to the user