Scoring (Vector Space Model)

CE-324: Modern Information Retrieval

Sharif University of Technology

M. Soleymani

Fall 2015

Most slides have been adapted from: Profs. Manning, Nayak & Raghavan (CS-276, Stanford)

Outline

- Ranked retrieval
- Scoring documents
  - Term frequency
  - Collection statistics
  - Term weighting
    - Weighting schemes
  - Vector space scoring

Ranked retrieval

- Boolean models:
  - Queries have all been Boolean.
  - Documents either match or don't.

- Boolean models are not good for the majority of users.
  - Most users incapable of writing Boolean queries.
    - a query language of operators and expressions
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.

Problem with Boolean search: feast or famine

- Too few (=0) or too many unranked results.
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

Ranked retrieval models

- Return an ordering over the (top) documents in the collection for a query
  - **Ranking** rather than a set of documents
  - **Free text queries**: query is just one or more words in a human language

- In practice, ranked retrieval has normally been associated with free text queries and vice versa

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - We just show the top k (≈ 10) results
  - We don't overwhelm the user
- Premise: the ranking algorithm works

Scoring as the basis of ranked retrieval

- Return in order the docs most likely to be useful to the searcher

- How can we rank-order docs in the collection with respect to a query?
  - Assign a score (e.g. in [0, 1]) to each document
    - measures how well doc and query "match"

Query-document matching scores

- Assigning a score to a query/document pair

- Start with a **one-term query**
  - Score 0 when query term does not occur in doc
  - More frequent query term in doc gets higher score

- We will look at a number of alternatives for this.

Binary term-document incidence matrix

Each doc is represented by a binary vector $\in \{0,1\}^{|V|}$

	Antony and Cleopatra	Julius Caesar	The 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	1	1	1	1
worser	1	0	1	1	1	0

Term-document count matrices

- Number of occurrences of a term in a document:
  - Each doc is a **count vector** $\in \mathbb{N}^{|V|}$ (a column below)

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
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
worser	2	0	1	1	1	0

Bag of words model

- Vector representation doesn't consider the ordering of words in a doc
  - ***John is quicker than Mary*** and ***Mary is quicker than John*** have the same vectors

- This is called the **bag of words** model.
  - "recovering" positional information later in this course.

- For now: bag of words model

Term frequency tf

- **Term frequency** $\mathrm{tf}_{t,d}$: the number of times that term *t* occurs in doc *d*.

- How to compute query-doc match scores using $\mathrm{tf}_{t,d}$?
  - Raw term frequency is not what we want:
    - A doc with tf=10 occurrence of a term is more relevant than a doc with tf=1.
      - But not 10 times more relevant.
  - Relevance does not increase proportionally with $\mathrm{tf}_{t,d}$.

frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d}\,, & tf_{t,d} > 0 \\ 0, & otherwise \end{cases}$$

- Example:
  - $0 \rightarrow 0$
  - $1 \rightarrow 1$
  - $2 \rightarrow 1.3$
  - $10 \rightarrow 2$
  - $1000 \rightarrow 4$

First idea

- Score for a doc-query pair (q, d_i):

$$score(q, d_i) = \sum_{t \in q} w_{t,i} = \sum_{t \in q \cap d_i} \left(1 + \log_{10} tf_{t,i}\right)$$

- It is 0 if none of the query terms is present in doc.

Term specificity

- Weighting the terms differently according to their specificity:
  - Term specificity is based on the accuracy of the term as a descriptor of a doc topic
  - It can be quantified as an inverse function of the number of docs in which occur (inverse doc frequency)

Document frequency

- Rare terms can be more informative than frequent terms
  - Stop words are not informative
  - frequent terms in the collection (e.g., *high, increase, line*)
    - A doc containing them is more likely to be relevant than a doc that doesn't
    - But it's not a sure indicator of relevance
      - High positive weights for such words
      - But lower weights than for rare terms
  - a query term that is rare in the collection (e.g., *arachnocentric*)
    - A doc containing it is very likely to be relevant to the query

- The most informative terms are nouns or noun groups

Document frequency (cont'd)

- Frequent terms are less informative than rare terms
  - We want a high weight for rare terms
- We will use doc frequency (df) to capture this.

Collection frequency vs. Doc frequency

- **Collection frequency of *t*:** number of occurrences of *t* in the collection, counting multiple occurrences.

- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- Which word is a better search term (and should get a higher weight)?

idf weight

- df_t (document frequency of t): the number of docs that contain *t*
  - df_t is an **inverse measure of informativeness** of *t*
  - $\mathrm{df}_t \leq N$
- idf (inverse document frequency of *t*)
  - log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

$$\mathrm{idf}_t = \log_{10} N/\mathrm{df}_t$$

Will turn out the base of the log is immaterial.

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	0

$$\mathrm{idf}_t = \log_{10} N/\mathrm{df}_t$$

There is one idf value for each term *t* in a collection.

idf example, suppose $N = 1$ million

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$\mathrm{idf}_t = \log_{10} N/\mathrm{df}_t$$

There is one idf value for each term t in a collection.

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries

- idf has no effect on ranking one term queries
  - affects for queries with at least two terms
  - Example query: capricious person
    - idf weighting makes occurrences of capricious count for much more in final doc ranking than occurrences of person.

TF-IDF weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.
  - Increases with number of occurrences within a doc
  - Increases with the rarity of the term in the collection

$$\mathrm{tf.idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

- Best known weighting scheme in information retrieval
  - Alternative names: tf.idf, tf x idf

TF-IDF weighting

- A common tf-idf:

$$w_{t,i} = \begin{cases} \left(1 + \log_{10} \mathrm{tf}_{t,i}\right) \times \log_{10} N/\mathrm{df}_t\,, & t \in d_i \\ 0, & otherwise \end{cases}$$

- Score for a document given a query via tf-idf:

$$score(q, d_i) = \sum_{t \in q} w_{t,i}$$

$$= \sum_{t \in q \cap d_i} \left(1 + \log_{10} \mathrm{tf}_{t,i}\right) \times \log_{10} N/\mathrm{df}_t$$

Document length normalization

- Doc sizes might vary widely

- Problem: Longer docs are more likely to be retrieved
- Solution: divide the rank of each doc by its length

- How to compute document lengths:
  - Number of words
  - Vector norms: $\|\vec{d}_j\| = \sqrt{\sum_{i=1}^{m} w_{i,j}^2}$

Documents as vectors

- $|V|$-dimensional vector space:
  - Terms are axes of the space
  - Docs are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions for a web search engine
- These are very sparse vectors (most entries are zero).

Binary → count → 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 doc is now represented by a real-valued vector ($\in \mathrm{R}^{|V|}$) of tf-idf weights

Queries as vectors

- Key idea 1: Represent docs also as vectors
- Key idea 2: Rank docs according to their proximity to the query in this space

- proximity = similarity of vectors
- proximity ≈ inverse of distance

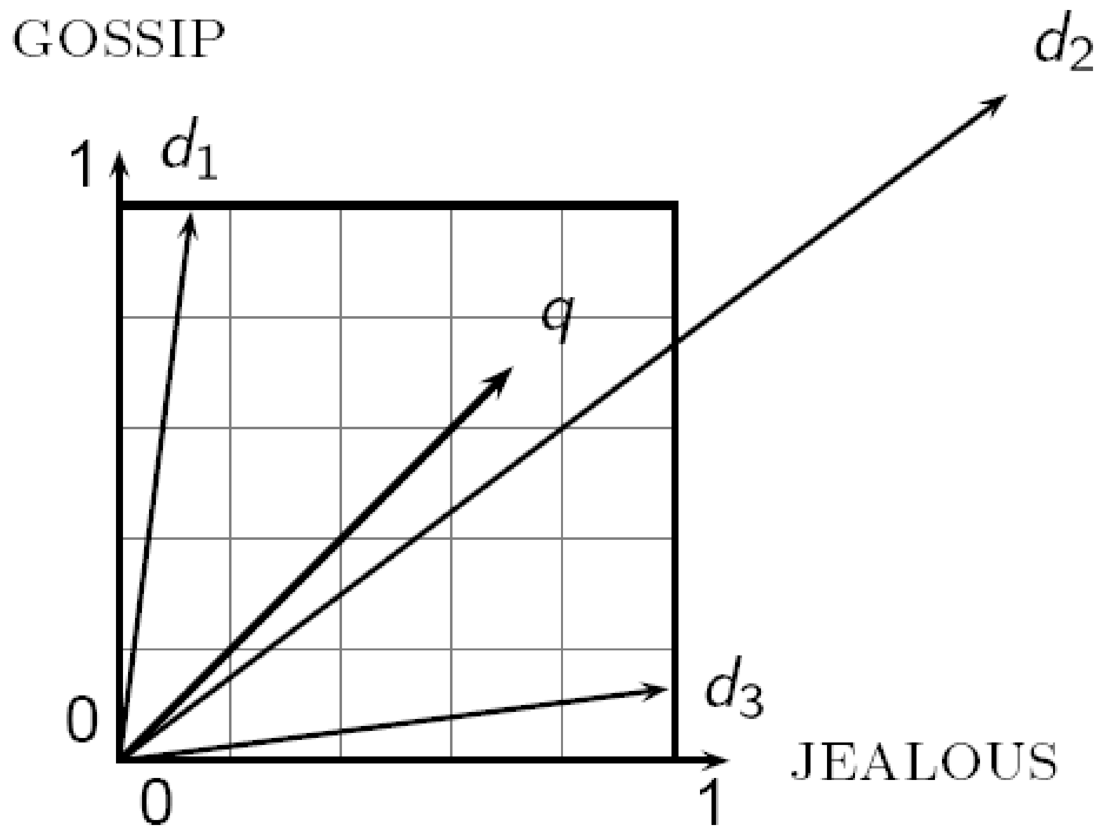
- To get away from you're-either-in-or-out Boolean model.
  - Instead: rank more relevant docs higher than less relevant docs

Formalizing vector space proximity

- First cut: distance between two points
  - distance between the end points of the two vectors
- Euclidean distance?
  - Euclidean distance is not a good idea . . .
    - It is large for vectors of different lengths.

Why distance is a bad idea

- Euclidean(q,d_2) is large
- While distribution of terms in q and d_2 are very similar.



Use angle instead of distance

- Experiment:
  - Take d and append it to itself. Call it d'.
  - "Semantically" d and d' have the same content
  - Euclidean distance between them can be quite large
  - Angle between them is 0, corresponding to maximal similarity.

- Key idea: Rank docs according to angle with query.

From angles to cosines

- The following two notions are equivalent.
  - Rank docs in decreasing order of the $\underline{angle(q, d)}$
  - Rank docs in increasing order of $\underline{cosine(q, d)}$

- Cosine is a monotonically decreasing function for the interval [0°, 180°]
  - But how – *and why* – should we be computing cosines?

Length normalization

- Length (L_2 norm) of vectors:

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

- (length-) normalized Vector: Dividing a vector by its length
  - Makes a unit (length) vector
  - Vector on surface of unit hypersphere

$$\frac{\vec{x}}{\|\vec{x}\|}$$

Length normalization

- d and d' (d appended to itself) have identical vectors after length-normalization.
  - Long and short docs now have comparable weights

Cosine similarity amongst 3 documents

- How similar are these novels?

  SaS: *Sense and Sensibility*

  PaP: *Pride and Prejudice*

  WH: *Wuthering Heights*

Term frequencies (counts)

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

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

cos(SaS,PaP) ≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

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

Cosine (query,document)

$$cos(\vec{d},\vec{q}) = \overbrace{\frac{\vec{d}.\vec{q}}{\|\vec{d}\|\|\vec{q}\|}}^{\text{Dot product}} = \overbrace{\frac{\vec{d}}{\|\vec{d}\|}.\frac{\vec{q}}{\|\vec{q}\|}}^{\text{Unit vectors}}$$

q_t : tf-idf weight of term t in query

d_t : tf-idf weight of term t in doc

$\cos(\vec{q},\vec{d})$: cosine similarity of *q* and *d*
(cosine of the angle between *q* and *d*.)

Cosine (query,document)

$$cos(\vec{d},\vec{q}) = \frac{\vec{d}.\vec{q}}{\|\vec{d}\|\|\vec{q}\|} = \frac{\vec{d}}{\|\vec{d}\|}.\frac{\vec{q}}{\|\vec{q}\|}$$

$$sim(d,q) = \frac{\vec{d}.\vec{q}}{\|\vec{d}\|\|\vec{q}\|} = \frac{\sum_{t=1}^{m} w_{t,d} \times w_{t,q}}{\sqrt{\sum_{t=1}^{m} w_{t,d}^2} \times \sqrt{\sum_{t=1}^{m} w_{t,q}^2}}$$

$\cos(\vec{q},\vec{d})$: cosine similarity of *q* and *d*
(cosine of the angle between *q* and *d*.)

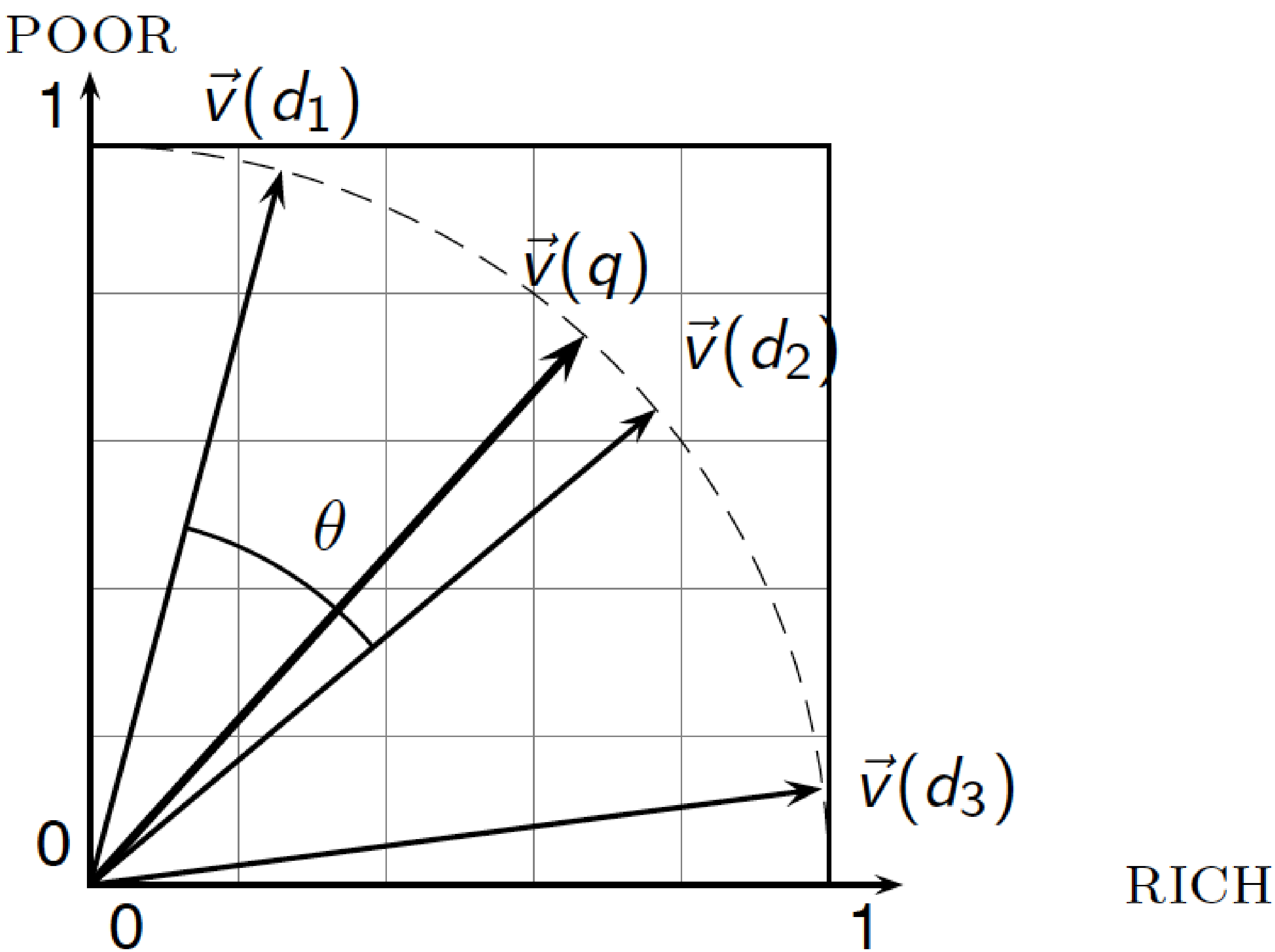
Cosine for length-normalized vectors

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

$$cos(\vec{d}, \vec{q}) = \frac{\vec{d}.\vec{q}}{\|\vec{d}\|\|\vec{q}\|} = \vec{d}.\vec{q}$$

for length-normalized q, d

Cosine similarity illustrated



Cosine similarity score

- A doc may have a high cosine score for a query even if it does not contain all query terms

- We use the inverted index to speed up the computation of the cosine score

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 $Scores[]$

tf-idf example: lnc.ltc

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

Term	Query						Document				Prod
	tf-raw						tf-raw				
auto	0						1				
best	1						0				
car	1						1				
insurance	1						2				

Exercise: what is *N*, the number of docs?

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

Score = 0+0+0.27+0.53 = 0.8

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

Exercise: what is *N*, the number of docs?

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

Score = 0+0+0.27+0.53 = 0.8

Variants of TF

Weighting scheme	TF weight
binary	$\{0,1\}$
raw frequency	$tf_{i,j}$
log normalization	$1 + \log\ tf_{i,j}$
double normalization 0.5	$0.5 + 0.5 \dfrac{tf_{i,j}}{\max_i\ tf_{i,j}}$
double normalization K	$K + (1 - K) \dfrac{tf_{i,j}}{\max_i\ tf_{i,j}}$

Variants of IDF

Weighting scheme	IDF weight
unary	1
inverse frequency	$\log \frac{N}{n_i}$
inverse frequency smooth	$\log \left(1 + \frac{N}{n_i}\right)$
inverse frequency max	$\log \left(1 + \frac{\max_i n_i}{n_i}\right)$
Probabilistic inverse frequency	$\log \frac{N - n_i}{n_i}$

TF-IDF weighting has many variants

Term frequency		Document frequency		Normalization		
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1	Default
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)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$					

Columns headed 'n' are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

Weighting may differ in queries vs docs

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

- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq*
  - A very standard weighting scheme is: lnc.ltc

ddd.qqq: example lnc.ltn

- Document:
  - l: logarithmic tf
  - n: no idf
  - c: cosine normalization

- Query:
  - l: logarithmic tf
  - t: idf (t in second column)
  - n: no normalization

Isn't it bad to not idf-weight the document?

Summary

- Represent the query as a weighted tf-idf vector
- Represent each doc as a weighted tf-idf vector
- Compute the similarity score of the query vector to doc vectors
  - May be different weighing for the query and docs
- Rank doc with respect to the query by score
- Return the top K (e.g., $K = 10$) to the user

Resources for today's lecture

- IIR 6.2 – 6.4.3
- MIR 3.2.3 – 3.2.6
- http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html
  - Term weighting and cosine similarity tutorial for SEO folk!