Information Retrieval: the Vector space model

COMP90042 Lecture 2

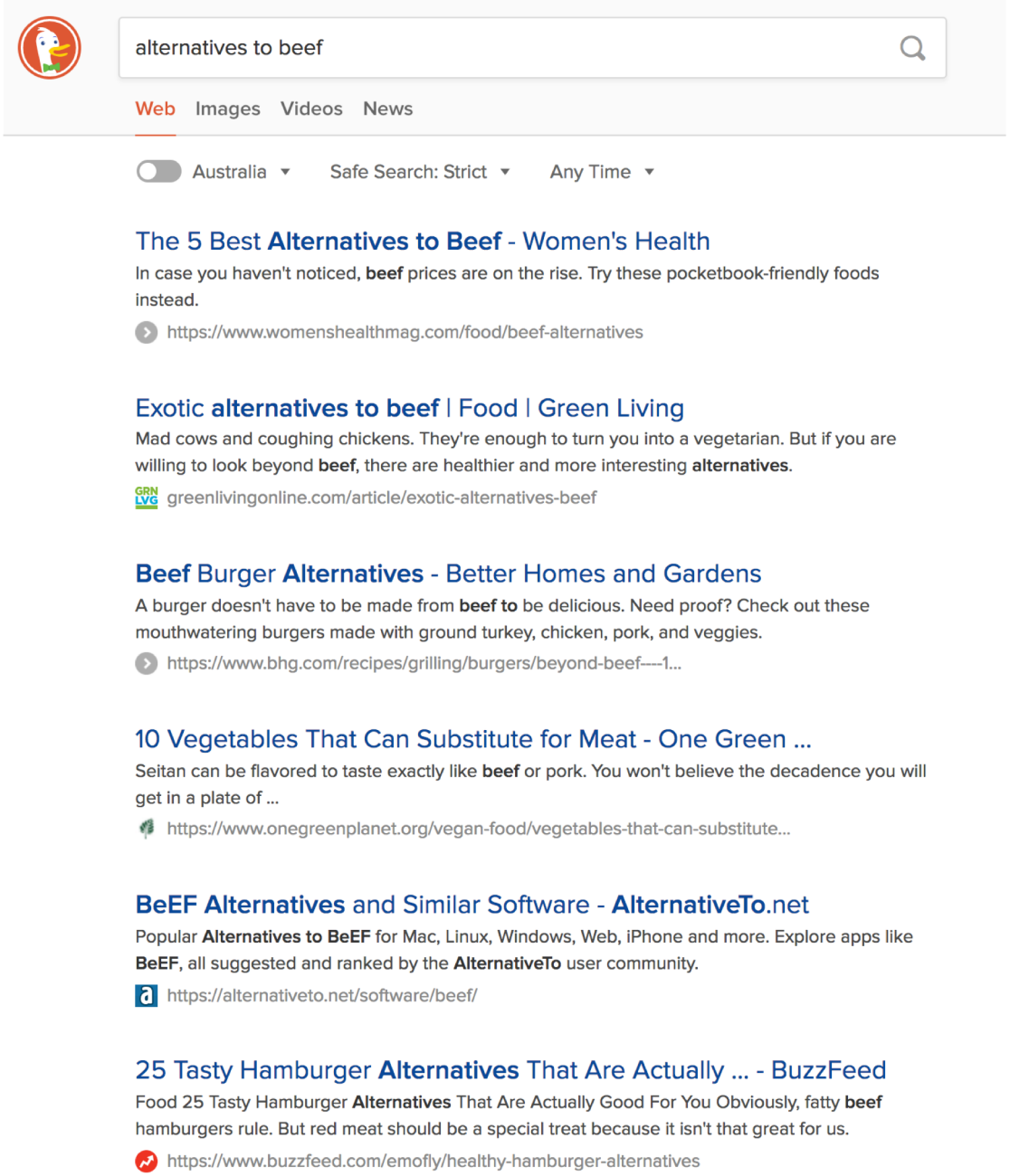
THE UNIVERSITY OF MELBOURNE

Overview

- Concepts of the *Term-Document Matrix* and *inverted index*
- Vector space measure of query-document similarity
- Efficient search for best documents

The problem of Info. retrieval

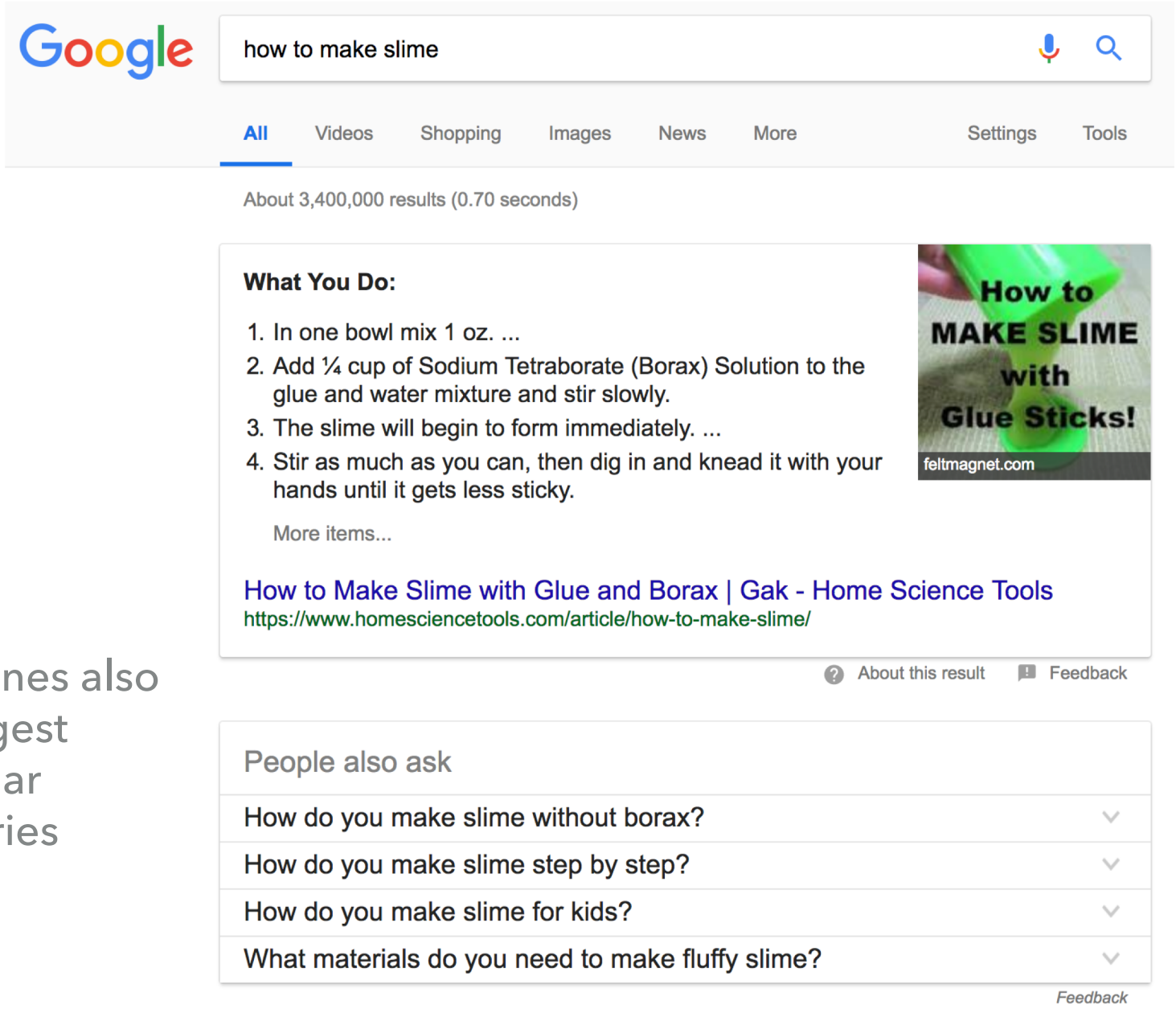
- Given a large document collection and a query string
  - which of the documents *are (most) relevant* to the user's information need?
- Information need ≠ query string
- Raises questions:
  - how to define query-document relevance?
  - how can we service queries efficiently?
  - how can we evaluate effectiveness?



What is the information need?

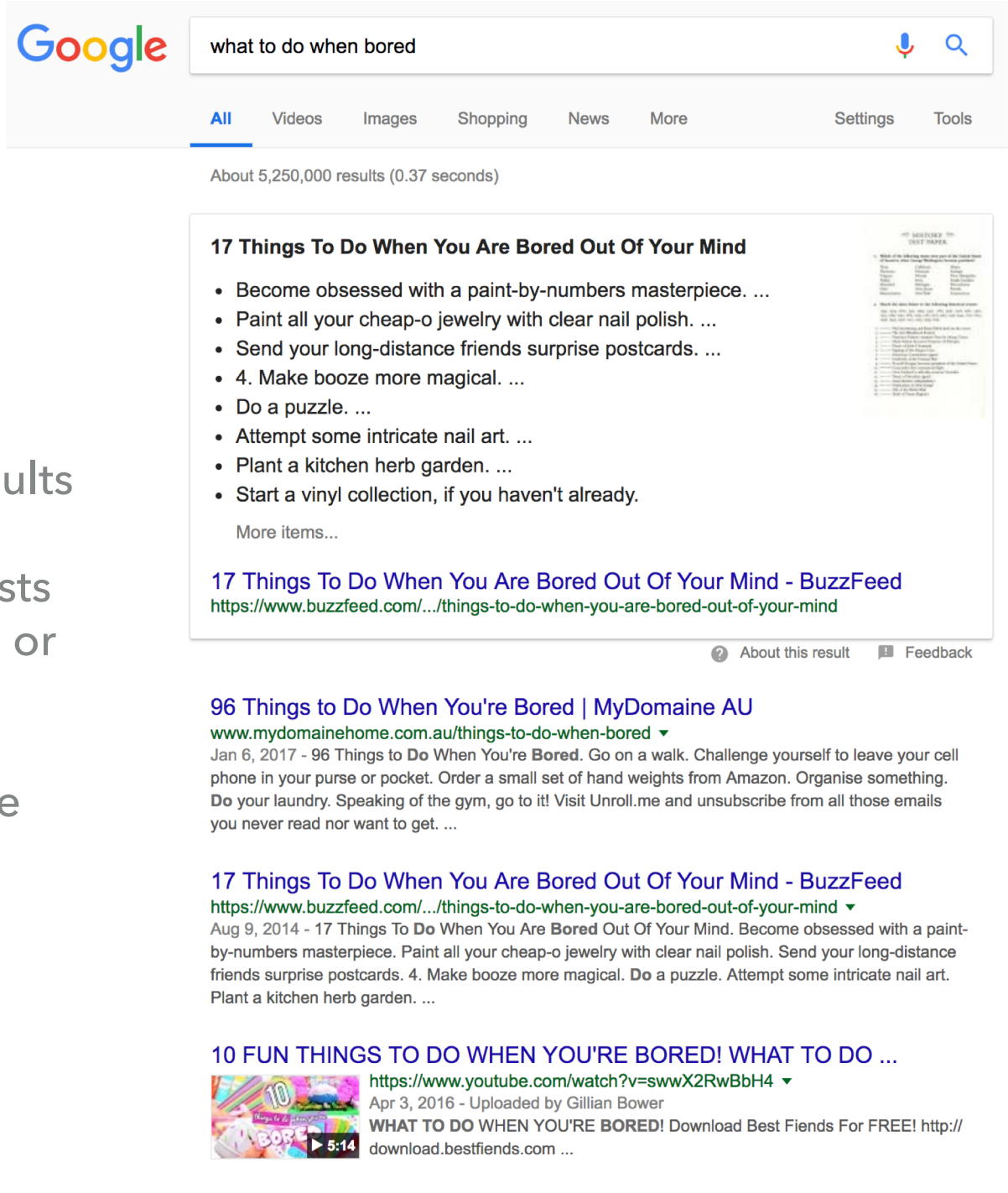
Does the user just want one answer or several?

Engine returns ranked answers with short snippets (*what underlies the ranking*?)



Engines also suggest similar queries

Often results might include lists of things, or many (diverse) results are desired



Evaluation on Test collections

- IR research often use reusable test collections constructed for *reproducible* IR evaluation, e.g., TREC competitions; comprising
  - ***corpus*** of documents
  - set of ***queries ("topics")***, often including long-form elaboration of *information need*
  - relevance judgements (***qrels***) for each document and query, a human judgement of whether the document is relevant to the information need in the given query.

Example from TREC 5

⟨num⟩ Number: 252
⟨title⟩ Topic: Combating Alien Smuggling
⟨desc⟩ Description: What steps are being taken by governmental or even private entities world-wide to stop the smuggling of aliens.
⟨narr⟩ Narrative: To be relevant, a document must describe an effort being made (other than routine border patrols) in any country of the world to prevent the illegal penetration of aliens across borders.

Qrels

Topic	Docid	Rel
252	AP881226-0140	1
252	AP881227-0083	0
252	CR93E-10038	0
252	CR93E-1004	0
252	CR93E-10211	0
252	CR93E-10529	1
...		

Runfile

Topic	Docid	Score
252	CR93H-9548	0.5436
252	CR93H-12789	0.4958
252	CR93H-10580	0.4633
252	CR93H-14389	0.4616
252	AP880828-0030	0.4523
252	CR93H-10986	0.4383
...		

Document representation

- Assume each document is a *bag of words,* i.e., discarding word order information, just recording counts
- The whole collection could be modelled as a *list of bag of words*
  - but this doesn't allow efficient access, e.g., to find a specific word
- Solution: the *term-document* matrix
  - rows represent *documents*
  - columns represent *terms* which are typically *word types* (excluding very high frequency "stop words", and often stemmed)
  - matrix cells might be binary indicators, frequency counts or some other kind of 'score' attached to a word and a document

Term-document matrix

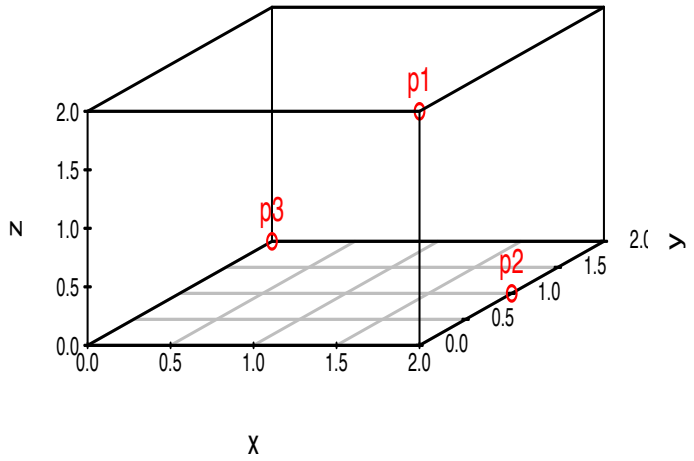
doc1	Two for tea and tea for two
doc2	Tea for me and tea for you
doc3	You for me and me for you

	two	tea	me	you
doc1	2	2	0	0
doc2	0	2	1	1
doc3	0	0	2	2

Geometrical view of the tdm

- The TDM not just a useful document representation
  - also suggests a useful way of modelling documents
  - consider documents as *points (vectors)* in a *multi-dimensional term space*

Point	x	y	z
p1	2	0	2
p2	2	1	0
p3	0	2	0

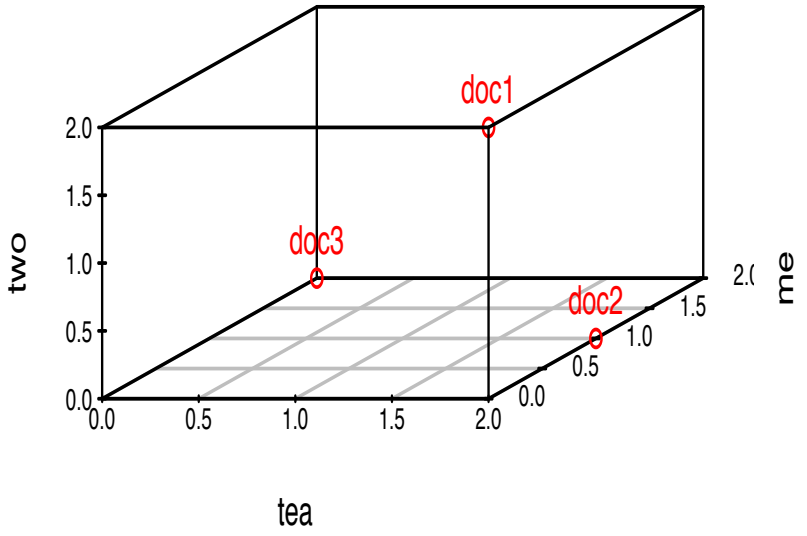


- E.g., points in 3d

Documents in term space

- Each of the |T| terms becomes a dimension
- Documents are points, with the value for each dimension determined by the score $s_{d,t}$
  - this might be the frequency of term *t* in document *d*

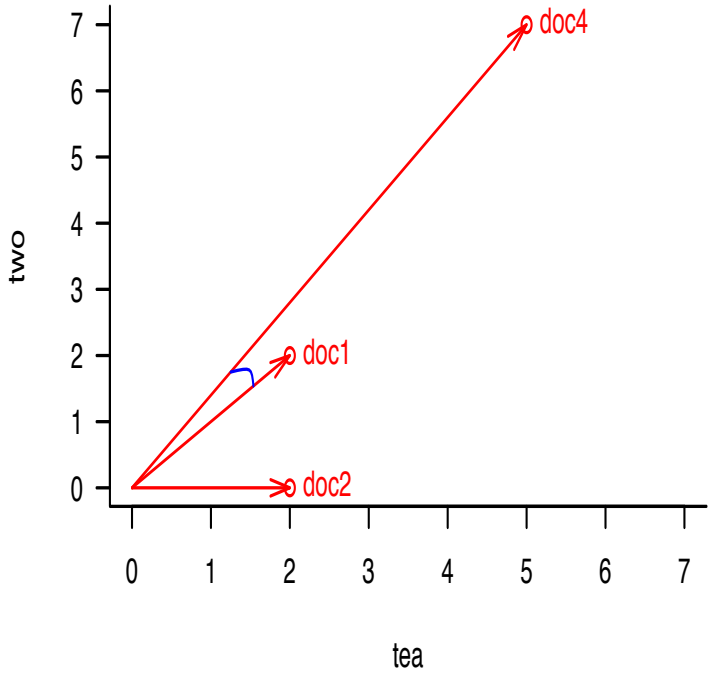
Point	tea	me	two
doc1	2	0	2
doc2	2	1	0
doc3	0	2	0



Measuring similarity with cosine

- To account for length bias consider documents as *vectors*
  - and measure the angle between them

Point	tea	two
doc1	2	2
doc2	2	0
doc4	5	7



cosine distance

- Given two vectors, **a** and **b**, the cosine between them is

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}.\mathbf{b}}{|\mathbf{a}| \times |\mathbf{b}|}$$

- where . is the vector dot product, i.e., $\mathbf{a}.\mathbf{b} = \sum_{i=1}^{|T|} a_i b_i$

- and |**a**| denotes the vector magnitude $|\mathbf{a}| = \sqrt{\sum_{i=1}^{|T|} a_i^2}$

Speeding up cosine distance

- All documents pre-normalised to unit length (or normalisation factors pre-computed and stored)
- Cosine then just requires a dot product **a**.**b**
  - but these vectors are sparse, with many 0 entries
  - only need to consider non-zero entries in both **a** and **b**
  - use inverted index to exploit sparsity, and allow efficient storage and querying
- This is known as *the vector space model*

TF*IDF

- Scores in TDM typically combine two elements, $tf_{d,t} \times idf_t$
  - The term frequency or TF, based on the occurrence count for the term in a document (e.g. raw count, log(1 + c),…)
  - The inverse document frequency or IDF, based on how rare the word is, and therefore how informative an occurrence will be
  - IDF typically formulated as $\text{idf}_t = \log \frac{N}{\text{df}_t}$ where N is the number of documents and df_t the number of documents containing t
- Consider extrema for idf, rare words vs. stop-words

TF x IDF example

- Raw TF

	two	tea	me	you
doc1	2	2	0	0
doc2	0	2	1	1
doc3	0	0	2	2

- IDF
  - "two" occurs in 1 doc (doc1)
    $idf_{two} = \log_2(3/1) = 1.58$
  - "tea" occurs in 2 docs (doc1, doc2)
    $idf_{tea} = \log_2(3/2) = 0.58$
  - …
  - what about a stop word? e.g., "for" is in all docs.

TF x IDF example

	two	tea	me	you
doc1	2	2	0	0
doc2	0	2	1	1
doc3	0	0	2	2

TF

idf	1.58	0.58	0.58	0.58

TF x IDF: scale each TF column (term) by its IDF score

	two	**tea**	**me**	**you**
doc1	3.17	1.16	0	0
doc2	0	1.16	0.58	0.58
doc3	0	0	1.16	1.16

Fixed mistake 6/3/19

Query processing in VSM

- Treat the query as a short pseudo-document
- Calculate the similarity between the query pseudo-document and each document in the collection
- Rank documents by decreasing similarity with cosine
- Return to user the top *k* ranked documents

Example

Corpus:

	two	tea	me	you
doc1	2	2	0	0
doc2	0	2	1	1
doc3	0	0	2	2

Query: **tea me**

	two	tea	me	you
query	0	1	1	0

Example: TDM

- Corpus represented as TF x IDF matrix

	two	tea	me	you
doc1	3.17	1.16	0	0
doc2	0	1.16	0.58	0.58
doc3	0	0	1.16	1.16

e.g., |doc1| = (3.17^2 + 1.16^2 + 0^2 + 0^2)^(0.5)

- Normalise TF x IDF

	two	tea	me	you
doc1	0.93	0.34	0	0
doc2	0	0.82	0.41	0.41
doc3	0	0	0.71	0.71

normalised matrix has row for doc/|doc1|

* divide each row by vector magnitude, |row|

Fixed mistake 6/3/19

Example: query similarity

normalised TDM

	two	tea	me	you
doc1	0.93	0.34	0	0
doc2	0	0.82	0.41	0.41
doc3	0	0	0.71	0.71

query as unit vector

	two	tea	me	you
q	0	0.71	0.71	0

- Compute dot product for each document
  - doc1.q = 0 * 0.93 + 0.71 * 0.34 + 0 * 0.71 + 0 * 0 = 0.24
  - doc2.q = 0 * 0 + 0.71 * 0.82 + 0.71 * 0.41 + 0 * 0.41 = 0.87
  - doc3.q = 0 * 0 + 0.71 * 0 + 0.71 *0.71 + 0 * 0.71 = 0.5
- Rank by cosine score: doc2 > doc3 > doc1

Fixed mistake 6/3/19

Observations

1. Elements of the vectors which are zero do not contribute to cosine calculation
   * Zeros common with real data and large vocabularies
   * Also true of other scoring functions, e.g., log (1+TF), TF*IDF, BM25
2. Enumerating all the documents is inefficient

Can we devise a way to find the most similar documents efficiently?

Beyond cosine: BM25

$$w_{td} = \left[\log \frac{N - f_t + 0.5}{f_t + 0.5}\right] \quad \text{(idf)}$$

$$\times \frac{(k_1 + 1) f_{d,t}}{k_1 \left((1 - b) + b \frac{L_d}{L_{ave}}\right) + f_{d,t}} \quad \text{(tf and doc. length)}$$

$$\times \frac{(k_3 + 1)\, f_{q,t}}{k_3 + f_{q,t}} \quad \text{(query tf)}$$

- Parameterised scoring formula: k_1, b, k_3 need to be tuned
  - defaults k_1 = 1.5, b = 0.5, k_3 = 0
- BM25 most widely used method in IR

Index

- Imagine we pre-compute the TF*IDF vectors for all documents, and their vector lengths
(for normalisation)
- These do not change from query to query, so save time by pre-calculating their values
- But still need to iterate over every document…

Term-wise processing

- When measuring document similarity, only terms occurring in both vectors contribute to cosine score.
- So with the query as a pseudo-document need only consider terms that are
  - in the query; and in the document
- If we can efficiently index the documents in which each term occurs
  - complexity reduced to $O(\sum_t \mathrm{df}_t)$
  - note that most frequent term will dominate (consider stop-words)

Inverted index

- Inverted index comprises
  - Terms as rows
  - Values as lists of (docID, weight) pairs, aka *posting list*

two	→	1: 0.88	
tea	→	1: 0.47	2: 0.9
me	→	2: 0.32	3: 0.71

  - weights listed are the normalised TF*IDF values (although may chose to store raw counts in practice)

Querying an inverted index

Assuming normalised TF*IDF weights:

Set accumulator $a_d \leftarrow 0$ for all documents d
for all terms t in query **do**
    Load postings list for t
    for all postings $\langle d, w_{t,d} \rangle$ in list **do**
        $a_d \leftarrow a_d + w_{t,d}$
    end for
end for
Sort documents by decreasing a_d
return sorted results to user

$w_{t,d}$ denotes normalised $tf_{d,t} \times idf_t$ vector

Example

tea	→	1: 0.47	2: 0.9
me	→	2: 0.32	3: 0.71

- a_d = < 0, 0, 0 >
- term ***"tea"***
  - $a_d[1]$ += 0.47
  - $a_d[2]$ += 0.9
  - end of block, a_d = < 0.47, 0.9, 0 >
- term ***"me"***
  - $a_d[2]$ += 0.31
  - $a_d[3]$ += 0.71
  - end of block, a_d = < 0.47, 1.21, 0.71 >
- sort to produce doc ranking
  - 2: 1.21; 3: 0.71; 1: 0.47 → [2, 3, 1]

Summary

- Concept of the term-document matrix and inverted index
- The vector space model, TF*IDF similarity and other scoring methods
- Inverted index & querying algorithm
- Reading
  - Chapter 6, “Scoring, term weighting & the vector space model” of Manning, Raghavan, and Schütze, Introduction to Information Retrieval
  - (opt.) Section 11.4.3 “Okapi BM25: A nonbinary model”