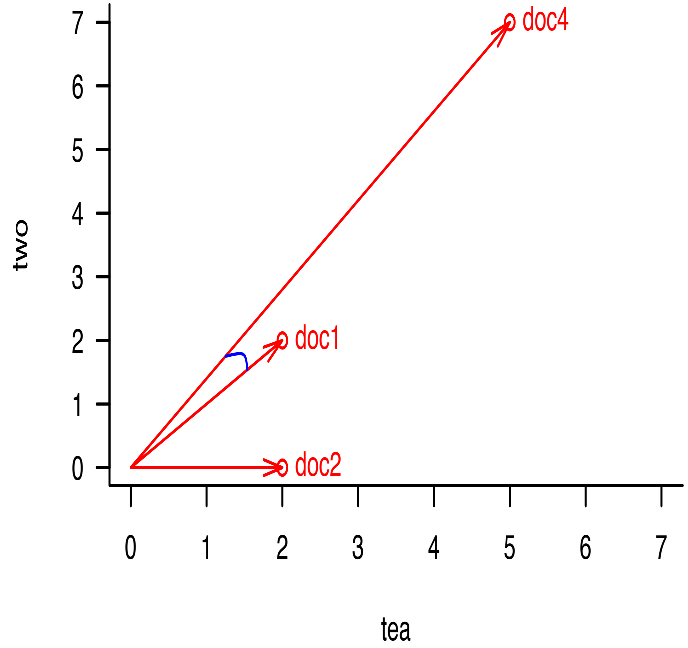

COMP90042 LECTURE 16

QUERYING THE VECTOR SPACE MODEL

OVERVIEW

- Taking a query and returning set of ranked results
- Efficient implementation
- Evaluation

RECAP: DOC. SIMILARITY IN VSM

- Document is a bag-of-words
- Project into term space as vector, with dimension lengths given by TF*IDF
- Calculate document similarity as cosine of angle between their vectors
- Implemented as dot product over unit-length vectors

Same process can be used to *rank* documents based on similarity to a given document

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:

Using raw term frequencies, and no IDF component

	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 (CONT) - NORMALISATION

Corpus:

	two	tea	me	you
doc1	0.707	0.707	0	0
doc2	0	0.816	0.408	0.408
doc3	0	0	0.707	0.707

Query: **tea me**

	two	tea	me	you
query	0	0.707	0.707	0

EXAMPLE

$$
\begin{aligned}
\cos(\text{doc1}, q) &= (0.707, 0.707, 0, 0) \cdot (0, 0.707, 0.707, 0) \\
&= 0.5 \\
\cos(\text{doc2}, q) &= (0, 0.816, 0.408, 0.408) \cdot (0, 0.707, 0.707, 0) \\
&= 0.866 \\
\cos(\text{doc3}, q) &= (0, 0, 0.707, 0.707) \cdot (0, 0.707, 0.707, 0) \\
&= 0.5
\end{aligned}
$$

- doc2 is the best match, followed by doc1 and doc3 (tied)

OBSERVATIONS

1. Many elements of the vectors were zero, and did not contribute to cosine calculation
   - Zeros common with real data and large vocabularies
   - Also true of other weightings, e.g., log TF and TF*IDF
2. Enumerating all the documents is inefficient

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

INDEX

- Imagine we precompute 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 precalculating 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*

Term		Postings list
tea	$\rightarrow$	1:1.4 ; 3:1.0 ; 6:1.7 ; ...
two	$\rightarrow$	2:2.3 ; 3:1.0 ; 4:1.7 ; ...
me	$\rightarrow$	1:1.0 ; 2:1.4 ; ...

  - weights listed might be normalised TF*IDF values, e.g.
- Note the inclusion of weight cf binary index

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

What about query magnitude?

EFFICIENT STORAGE OF INV. INDEX

- Index can be very large; seek to optimise memory footprint
  - in order to fit in memory, or compactly on disk
- Size implications of design choices
  - integer values (counts) can be easily compressed, less easy for real values
  - may not want to store TF*IDF values and normalised vectors
- Instead record separately:
  - raw count data in postings lists;
  - document frequency for each term; and
  - document length normalisation values.

EFFICIENT INDEX

Inverted index mostly comprised of integer counts

Term	IDF		Postings list
tea	1.9	$\rightarrow$	1:3 ; 3:1 ; 6:2 ; ...
two	0.3	$\rightarrow$	2:4 ; 3:1 ; 4:2 ; ...
me	0.8	$\rightarrow$	1:1 ; 2:2 ; ...

And real valued document length

DocId	$\lvert w_{\cdot,d} \rvert$
1	2.3
2	3.4
3	1.7
4	42.8
$\vdots$	$\vdots$

QUERYING IN SPACE EFFICIENT INDEX

Set accumulator $a_d \leftarrow 0$ for all documents d
for all terms t in query **do**
    Load postings list for t and $\text{idf}_t = \log \frac{N}{f_t}$
    for all postings $\langle d, f_{t,d} \rangle$ in list **do**
        $a_d \leftarrow a_d + f_{t,d} \times \text{idf}_t$
    end for
end for
Load document length array, L
Normalise by document lengths $a_d \leftarrow \frac{a_d}{L_d}$
Sort documents by decreasing a_d
return sorted results to user

- A little more computation in inner loop, but supports more compact storage.

HOW IS THIS COSINE?

- The algorithm computes for each document

$$a_d = \frac{\sum_{t \in q} w_{t,d}}{\sqrt{\sum_{t \in d} w_{t,d}^2}}$$

- But cosine is defined as

$$\cos(d, q) = \frac{\sum_t w_{t,q} w_{t,d}}{\sqrt{\sum_{t \in q} w_{t,q}^2 \sum_{t \in d} w_{t,d}^2}}$$

- What happened to the query term-weights and normalisation term?

HOW IS THIS COSINE?

- Assume that each query term occurs once in the query
  - $w_{t,q}$ = 1 for all t in the query (and 0 for the remaining terms)
- The query length is irrelevant
  - compare one fixed query to several documents
  - scaling by a constant (query length) means ranking remains the same

EVALUATING EFFECTIVENESS

- Hard to characterise the quality of a system's results
  - a subjective problem, depends on the user's information need and how well the results meet that need
  - query is not the information need itself, but an expression thereof
- Obvious evaluation method: human judgements
  - directly measure effectiveness in user studies; for reported satisfaction, completion of tasks, …
  - but too expensive and slow, especially when tuning parameters of the system (e.g., flavour of TF*IDF, use of stopwords, etc…)

AUTOMATIC EVALUATION

- Make simplifying assumptions
  - retrieval is ad-hoc
    - query performed only once, and with no prior knowledge of the user or their behavior
  - effectiveness based on relevance
    - each document is either relevant or irrelevant to information need (binary)
    - relevance of documents are independent from others (no consideration of redundancy)
- Effectiveness is a function of the relevance of documents returned by the system

TEST COLLECTIONS

- Several reusable test collections constructed for IR evaluation, e.g., for TREC competitions; comprising
  - ***corpus*** of documents
  - set of ***queries***, sometimes 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.
- Typically not all documents have *qrels,* collection is simply too big and most are likely to be irrelevant.

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
...		

EXAMPLE RELEVANCE VECTOR

- Based on retrieval run, calculate binary vector indicating relevance for each ranked document

Retrieval run

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

Qrels

Docid	Rel
AP880828-0030	0
AP881226-0140	1
AP881227-0083	0
CR93H-14389	0
CR93H-9548	1
CR93H-10580	0
CR93H-10986	1
CR93H-12789	0
...	
...	

Relevance vector

$$\langle 1, 0, 0, 0, 0, 1, \ldots \rangle$$

RELEVANCE MEASURES

- How to map relevance vector to a number?
- Natural candidates are precision & recall
  - but recall is hard to calculate (why?); and
  - how to deal with ranked outputs?
- Mainly use precision oriented metrics:
  - **precision @ k**: compute precision using only ranks 1 .. k
  - (mean) **average precision**: take average over prec@k for various k values; measure becomes *rank sensitive*
  - **mean reciprocal rank** (mrr): average over the *ranks* of relevant docs

RELEVANCE EXAMPLE

- Relevance vector

  `< 1, 0, 0, 0, 0, 1, 0, 1, 0, 0 >`

- Precision
  - P@1 = 1/1 P@2 = ½ P@3 = 1/3 P@4 = ¼
    P@5 = 1/5 P@6 = 2/6 P@7 = 2/7 P@8 = 3/8
    P@9 = 3/9 P@10 = 3/10
- AveP = 1/10 * (sum of above) = 0.391
- MRR = 1/10 * (1/1 + 1/6 + 1/8) = 0.129

Results then averaged over all queries in test collection.

SUMMARY

- Queries can be processed in VSM by treating as a pseudo-document
- Inverted index supports efficient query processing
- Evaluation using relevance judgements
- Precision@k, (M)AP, MRR evaluation metrics
- Reading
  - MRS Chapter 6.3
  - MRS Chapter 7.1