Information Retrieval

hussein suleman
uct cs honours 2008

Introduction

- Information retrieval is the process of locating the most relevant information to satisfy a specific information need.
- Traditionally, we used databases and keywords to locate information.

- The most common modern application is search engines.
- Historically, the technology has been developed from the mid-50’s onwards, with a lot of fundamental research conducted pre-Internet!

Terminology

- Term
  - Individual word, or possibly phrase, from a document.
- Document
  - Set of terms, usually identified by a document identifier (e.g., filename).
- Query
  - Set of terms (and other semantics) that are a machine representation of the user’s needs.
- Relevance
  - Whether or not a given document matches a given query.

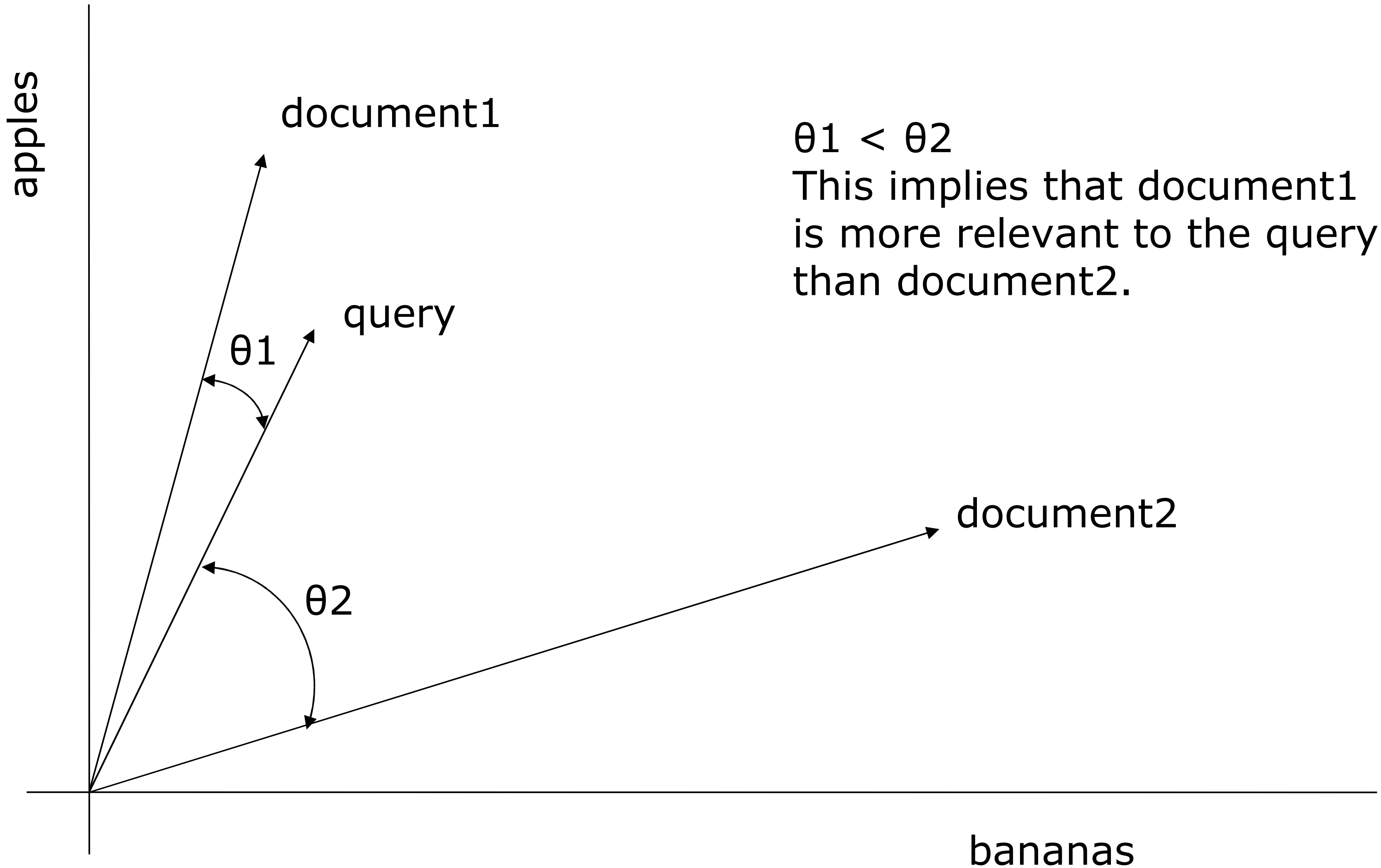
More Terminology

- Searching/Querying
  - Retrieving all the possibly relevant results for a given query.
- Indexing
  - Creating indices of all the documents/data to enable faster searching/quering.
- Ranked retrieval
  - Retrieval of a set of matching documents in decreasing order of estimated relevance to the query.

Models for IR

- Boolean model
  - Queries are specified as boolean expressions and only documents matching those criteria are returned.
    - e.g., apples AND bananas
- Vector model
  - Both queries and documents are specified as lists of terms and mapped into an *n*-dimensional space (where *n* is the number of possible terms). The relevance then depends on the angle between the vectors.

Vector Model in 2-D



θ1 < θ2
This implies that document1 is more relevant to the query than document2.

Extended Boolean Models

- Any modern search engine that returns no results for a very long query probably uses some form of boolean model!
  - Altavista, Google, etc.
  - Vector models are not as efficient as boolean models.
- Some extended boolean models filter on the basis of boolean matching and rank on the basis of term weights (tf.idf).

Filtering and Ranking

- Filtering
  - Removal of non-relevant results.
  - Filtering restricts the number of results to those that are probably relevant.
- Ranking
  - Ordering of results according to calculated probability of relevance.
  - Ranking puts the most probably relevant results at the "top of the list".

Efficient Ranking

- Comparing every document to each query is very slow.
- Use inverted files to speed up ranking algorithms by possibly ignoring:
  - terms with zero occurrence in each document.
  - documents where terms have a very low occurrence value.
- We are only interested in those documents that contain the terms in the query.

Inverted (Postings) Files

- An inverted file for a term contains a list of document identifiers that correspond to that term.

Doc1	apples bananas apples apples
Doc2	bananas bananas apples bananas bananas

original documents

inverted files

apples	Doc1: 3 Doc2: 1	4
bananas	Doc1: 1 Doc2: 4	5

Implementation of Inverted Files

- Each term corresponds to a list of weighted document identifiers.
  - Each term can be a separate file, sorted by weight.
  - Terms, documents identifiers and weights can be stored in an indexed database.
- Search engine indices can easily take 2-6 times as much space as the original data.
  - The MG system (part of Greenstone) uses index compression and claims 1/3 as much space as the original data.

Inverted File Optimisations

- Use identifier hash/lookup table:
  - apples: 1 3 2 1
  - bananas: 1 1 2 4
- Sort weights and use differential values:
  - apples: 2 1 1 2
  - bananas: 1 1 2 3
- Aim: reduce values as much as possible so that optimal variable-length encoding schemes can be applied.
  - (For more information, read up on basic encoding schemes in data compression)

IF Optimisation Example

Id	W
1	3
2	2
3	7
4	5
5	1

Original inverted file

Sort on W(eight) column →

Id	W
5	1
2	2
1	3
4	5
3	7

Subtract each weight from the previous value →

Id	W'
5	1
2	1
1	1
4	2
3	2

Transformed inverted file – this is what is encoded and stored

Note: We can do this with the ID column instead!

To get the original data:
W[1] = W'[1]
W[i] = W[i-1]+W'[i]

Id	W
5	1
2	2
1	3
4	5
3	7

Boolean Ranking

- Assume a document D and a query Q are both *n*-term vectors.
- Then the inner product is a measure of how well D matches Q:

$$Similarity = D \cdot Q = \sum_{t=1}^{n} d_t . q_t$$

- Normalise so that long vectors do not adversely affect the ranking.

$$Similarity = \frac{1}{|D||Q|} \sum_{t=1}^{n} d_t . q_t$$

Boolean Ranking Example

- Suppose we have the document vectors D1:(1, 1, 0) and D2:(4, 0, 1) and the query (1, 1, 0).
- Non-normalised ranking:
  - D1: (1, 1, 0)·(1, 1, 0) = 1.1 + 1.1 + 0.0 = 2
  - D2: (4, 0, 1)·(1, 1, 0) = 4.1 + 0.1 + 1.0 = 4
  - Ranking: D2, D1
- Normalised ranking:

$$|D١| = \sqrt{\sum_{i=١}^{m} d_{١,i}^{٢}} = \sqrt{١.١ + ١.١ + ٠.٠} = \sqrt{٢} \qquad |D٢| = \sqrt{\sum_{i=١}^{m} d_{٢,i}^{٢}} = \sqrt{٤.٤ + ٠.٠ + ١.١} = \sqrt{١٧}$$

$$|Q| = \sqrt{\sum_{i=١}^{m} q_{i}^{٢}} = \sqrt{١.١ + ١.١ + ٠.٠} = \sqrt{٢}$$

- D1: (1, 1, 0)·(1, 1, 0)/√2.√2 = (1.1 + 1.1 + 0.0)/2 = 1
- D2: (4, 0, 1)·(1, 1, 0)/√17.√2 = (4.1 + 0.1 + 1.0)/√34 = 4/√34
- Ranking: D1, D2

tf.idf

- Term frequency (tf)
  - The number of occurrences of a term in a document – terms which occur more often in a document have higher tf.
- Document frequency (df)
  - The number of documents a term occurs in – popular terms have a higher df.
- In general, terms with high “tf” and low “df” are good at describing a document and discriminating it from other documents – hence tf.idf (term frequency * inverse document frequency).

Inverse Document Frequency

- Common formulation:

$$w_t = \log_e\left(1 + \frac{N}{f_t}\right)$$

- Where f_t is the number of documents term *t* occurs in (document frequency) and N is the total number of documents.
- Many different formulae exist – all increase the importance of rare terms.
- Now, weight the query in the ranking formula to include an IDF with the TF.

$$Similarity = \frac{1}{|D||Q|}\sum_{t=1}^{n} d_t . \log_e\left(1 + \frac{N}{f_t}\right)$$

Term Frequency

- Scale term frequency so that the subsequent occurrences have a lesser effect than earlier occurrences.
- Choose only terms in Q - as this is boolean - so prevent every term having a value of at least 1 (where before they were 0).

$$Similarity = \frac{١}{|D||Q|} \sum_{t \in Q \cap D} (١ + \log_e f_{d,t}).\log_e\left(١ + \frac{N}{f_t}\right)$$

- Lastly, eliminate |Q| since it is constant.

$$Similarity = \frac{١}{|D|} \sum_{t \in Q \cap D} (١ + \log_e f_{d,t}).\log_e\left(١ + \frac{N}{f_t}\right)$$

Vector Ranking

- In *n*-dimensional Euclidean space, the angle between two vectors is given by:

$$\cos\theta = \frac{X \cdot Y}{|X||Y|}$$

- Note:
  - cos 90 = 0 (orthogonal vectors shouldn't match)
  - cos 0 = 1 (corresponding vectors have a perfect match)
- Cosine θ is therefore a good measure of similarity of vectors.
- Substituting good tf and idf formulae in X.Y, we then get a similar formula to before (except we use all terms t[1..N].

Term Document Space

- A popular view of inverted files is as a matrix of terms and documents.

documents

terms

	Doc1	Doc2
Apples	3	1
Bananas	1	4

Clustering

- In term-document space, documents that are similar will have vectors that are "close together".
- Even if a specific term of a query does not match a specific document, the clustering effect will compensate.
- Centroids of the clusters can be used as cluster summaries.
- Explicit clustering can be used to reduce the amount of information in T-D space.

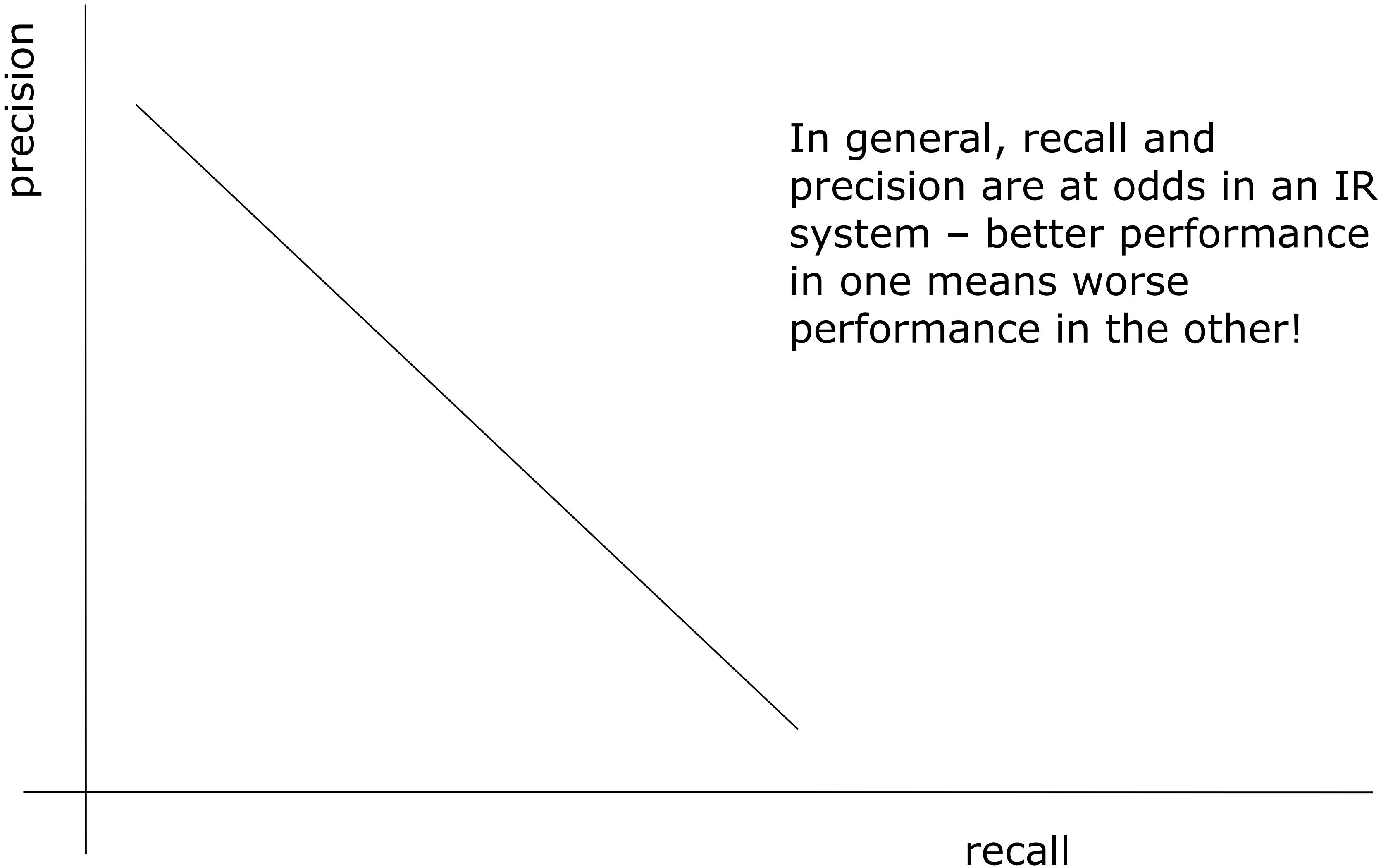
Evaluation of Retrieval Algorithms

- Recall
  - The number of relevant results returned.
  - Recall = number retrieved and relevant / total number relevant

- Precision
  - The number of returned results that are relevant.
  - Precision = number retrieved and relevant / total number retrieved

- Relevance is determined by an "expert" in recall/ precision experiments. High recall and high precision are desirable.

Typical Recall-Precision Graph



In general, recall and precision are at odds in an IR system – better performance in one means worse performance in the other!

Other Techniques to Improve IR

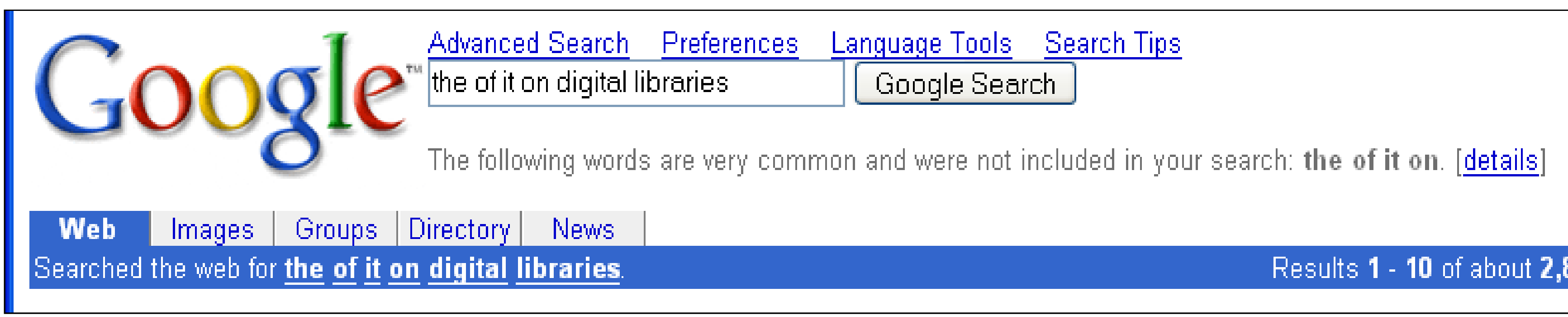
- Stemming, Stopping
- Thesauri
- Metadata vs. Fulltext
- Relevance Feedback
- Inference Engines
- LSI
- PageRank
- HITS

Stemming and Case Folding

- Case Folding
  - Changing all terms to a standard case, e.g., lowercase
- Stemming
  - Changing all term forms to canonical versions.
    - e.g., studying, studies and study map to “study”.
  - Stemming must avoid mapping words with different roots to the same term.
  - Porter’s Stemming Algorithm for English applies a set of rules based on patterns of vowel-consonant transitions.

Stopping

- Stopwords are common words that do not help in discriminating in terms of relevance.
  - E.g., in for the a an of on
- Stopwords are not standard and depend on application and language.



Thesauri

- A thesaurus is a collection of words and their synonyms.
  - e.g., According to Merriam-Webster, the synonyms for “library” are “archive” and “athenaeum”.
- An IR system can include all synonyms of a word to increase recall, but at a lower precision.
- Thesauri can also be used for cross-language retrieval.

Metadata vs. Full-text

- Text documents can be indexed by their contents or by their metadata.
- Metadata indexing is faster and uses less storage.
- Metadata can be obtained more easily (e.g., using open standards) while full text is often restricted.
- Full-text indexing does not rely on good quality metadata and can find very specific pieces of information.

Relevance Feedback

- After obtaining results, a user can specify that a given document is relevant or non-relevant.
- Terms that describe a (non-)relevant document can then be used to refine the query – an automatic summary of a document is usually better at describing the content than a user.

AltaVista found 825,158 results About

Libweb - Library WWW Servers
A global directory of library home pages ... type, name or other information. United States Academic **Libraries** Public **Libraries** National **Libraries** and Library Organizations State **Libraries** Regional ...
sunsite.berkeley.edu/Libweb • Refreshed in past 48 hours • Related Pages
More pages from sunsite.berkeley.edu

Inference Engines

- Machine learning can be used to digest a document collection and perform query matching.
  - Connectionist models (e.g., neural networks)
  - Decision trees (e.g., C5)
- Combined with traditional statistical approaches, this can result in increased recall/precision.

Latent Semantic Indexing

- LSI is a technique to reduce the dimensionality of the term-document space, resulting in greater speed and arguably better results.
- Problems with traditional approach:
  - Synonymy – two different words that mean the same thing.
  - Polysemy – two different meanings for a single word.
- LSI addresses both of these problems by transforming data to its "latent semantics."

Singular Value Decomposition

- SVD is used in LSI to factor the term-document matrix into constituents.
  - Calculations are based on eigenvalues and eigenvectors - many Mathematics packages can compute an SVD as a built-in function.

$$A = U \sum V^T = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \begin{bmatrix} * & & \\ & * & \\ & & * \\ & & \\ & & \end{bmatrix} \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$$

SVD Sizes

- If A, the term-document matrix, is an mxn matrix,
  - U is an mxm orthogonal matrix
  - V is an nxn orthogonal matrix
  - ∑ is the mxn diagonal matrix containing values on its diagonal in decreasing order of value. i.e., $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq \sigma_{\min(m,n)}$
- Note:
  - m is the number of terms, represented by the rows of A
  - n is the number of documents, represented by the columns of A

Approximation

- Replace ∑ with an approximation where the smallest values are zero.

$$\Sigma = \begin{bmatrix} \text{١.٥٧٨} & & & & \\ & \text{١.٣٢٠} & & & \\ & & \text{١.١١١} & & \\ & & & \text{٠.٨٧٠} & \\ & & & & \text{٠.٢٣٠} \end{bmatrix}$$

becomes,

$$\Sigma' = \begin{bmatrix} \text{١.٥٧٨} & & & & \\ & \text{١.٣٢٠} & & & \\ & & \text{١.١١١} & & \\ & & & \text{٠.٠} & \\ & & & & \text{٠.٠} \end{bmatrix}$$

Effect of Approximation

$$A' = U' \Sigma' {V^T}' = \begin{bmatrix} * & * & . & . & . \\ * & * & . & . & . \\ * & * & . & . & . \\ * & * & . & . & . \\ * & * & . & . & . \end{bmatrix} \begin{bmatrix} * & & \\ & * & \\ & & . \\ & & \\ & & \end{bmatrix} \begin{bmatrix} * & * & * \\ * & * & * \\ . & . & . \end{bmatrix}$$

- If only p values are retained in ∑, then only p columns of U and p rows of V must be stored.

LSI Example 1/2

- Consider a document collection:
  - D1: apples bananas bananas bananas pears
  - D2: bananas bananas bananas
  - D3: pears
- With query: q="apples"
- The term-document matrix will be:

	D1	D2	D3
apples	1	0	0
bananas	3	3	0
pears	1	0	1

LSI Example 2/3

svd.nb * -STUDENT VERSION-

Specify the term-document matrix.

In[1]:= $A = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 3 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

Out[1]= {{1, 0, 0}, {3, 3, 0}, {1, 0, 1}}

Create a query vector.

In[2]:= q = {1, 0, 0}

Out[2]= {1, 0, 0}

Find the inner product of the query with every original document (column of A) by matrix/vector multiplication.

In[3]:= Transpose[A] . q

Out[3]= {1, 0, 0}

Calculate the Singular Value Decomposition of A into three matrices.

In[4]:= {U, S, V} = SingularValues[N[A]]

Out[4]= {{{-0.170644, -0.968737, -0.180079}, {0.34885, -0.23032, 0.908436},
{-0.921512, 0.0921986, 0.377247}}, {4.36877, 1.27413, 0.53895},
{{-0.745504, -0.665225, -0.0412197}, {0.44448, -0.5423, 0.712985},
{-0.496649, 0.513212, 0.699966}}}

LSI Example 3/3

svd.nb * -STUDENT VERSION-

Check that the SVD, when multiplied, gives back the original matrix.

```
In[5]:= Transpose[U] . DiagonalMatrix[S] . V

Out[5]= {{1., -3.88578 × 10^-16, 3.33067 × 10^-16}, {3., 3., -2.498 × 10^-16},
         {1., 2.498 × 10^-16, 1.}}
```

Create a truncated S diagonal matrix, with all elements being 0 except the first.

```
In[6]:= Sp = Table[If[i == 1, S[[i]], 0], {i, 1, 3}]

Out[6]= {4.36877, 0, 0}
```

Multiply the new S with U and V, to create an approximation of the term-document space.

```
In[7]:= Ap = Transpose[U] . DiagonalMatrix[Sp] . V

Out[7]= {{0.555777, 0.495928, 0.0307295}, {3.15511, 2.81536, 0.174449},
         {0.586506, 0.523348, 0.0324285}}
```

Find the inner product of the query with every document of Ap, the approximated term-document space.

```
In[8]:= Transpose[Ap] . q

Out[8]= {0.555777, 0.495928, 0.0307295}
```

Note: in practice, LSI does not generate the approximated matrix.

Advantages of LSI

- Smaller vectors and pre-calculations result in faster query matching.
- Smaller term-document space – less storage required.
- Automatic clustering of documents based on mathematical similarity (basis vector calculations).
- Elimination of “noise” in document collection.

Web Data Retrieval

- Web crawlers are often bundled with search engines to obtain data from the WWW.
- Crawlers follow each link (respecting robots.txt exclusions) in a hypertext document, obtaining an ever-expanding collection of data for indexing/querying.
- WWW search engines operate as follows:

PageRank

- PageRank (popularised by Google) determines the rank of a document based on the number of documents that point to it, implying that it is an "authority" on a topic.
- In a highly connected network of documents with lots of links, this works well. In a diverse collection of separate documents, this will not work.
- Google uses other techniques as well!

Simple PageRank

- PageRank works with a complete collection of linked documents.
- Pages are deemed important if
  - They are pointed to by many other pages,
  - Each also of high importance.
- Define
  - r(i) = rank of a page
  - B(i)= set of pages that point to i
  - N(i) = number of pages that i points to

$$r(i) = \sum_{j \in B(i)} r(j) / N(j)$$

- Interpretation: r(j) distributes its weight evenly to all its N(j) children

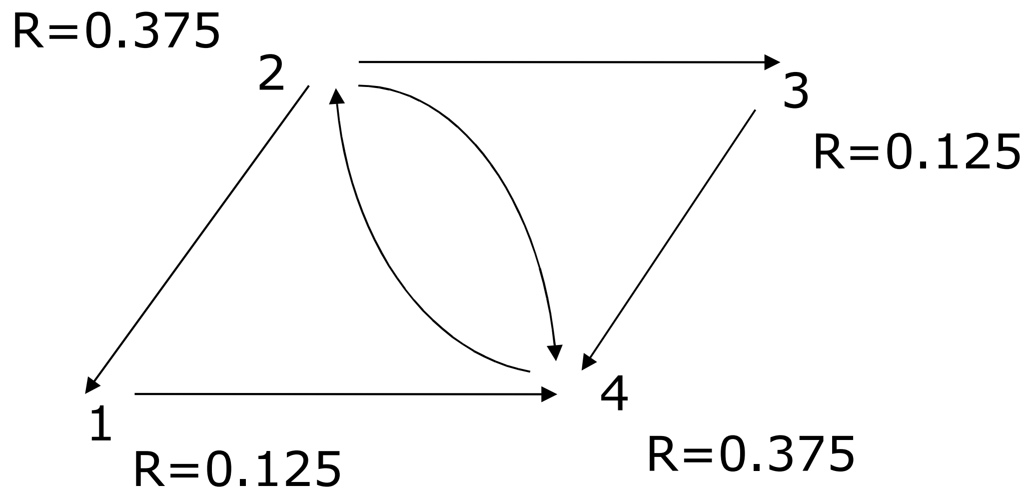
Computing PageRank

- Choose a random set of ranks and iterate until the relative order doesn’t change.

- Basic Algorithm:
  - s = random vector
  - Compute new $r(i)$ for each node
  - If $|r-s|<\varepsilon$, r is the PageRank vector
  - s = r, and iterate.

PageRank Example

Node	B(i)	N(i)
1	2	1
2	4	3
3	2	1
4	123	1



Node	$r_0(i)$	$r_1(i)$	$r_2(i)$	$r_3(i)$	…	$r_{200}(i)$
1	0.25	0.083	0.083	0.194	…	0.125
2	0.25	0.25	0.583	0.25	…	0.375
3	0.25	0.083	0.083	0.194	…	0.125
4	0.25	0.583	0.25	0.361	…	0.375

Sinks and Leaks

- In practice, some pages have no outgoing or incoming links.
- A “rank sink” is a set of connected pages with no outgoing links.
- A “rank leak” is a single page with no outgoing link.
- PageRank does the following:
  - Remove all leak nodes.
  - Introduce random perturbations into the iterative algorithm.

HITS

- Hypertext Induced Topic Search ranks the results of an IR query based on authorities and hubs.
- An authority is a page that many pages (hubs) point to.
  - E.g., www.uct.ac.za
- A hub is a page that points to many pages (authorities).
  - E.g., yahoo.com

HITS Algorithm 1/2

- Submit the query to an IR system and get a list of results.

- Create a focused subgraph as follows:
  - Let R = set of all result pages
  - Let S = R
  - Let Q = {}
  - For each page p in R
    - Add to Q all pages in S that p points to
    - Add to Q all pages (up to a limit) in S that point to p

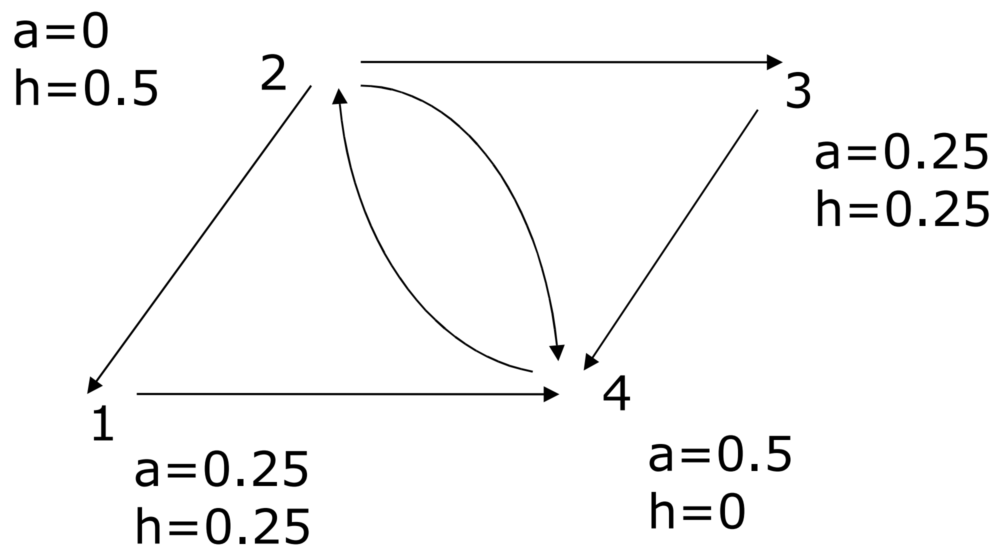
HITS Algorithm 2/2

- Initialise a_i and h_i for each node i to arbitrary values.

- Repeat until convergence:
  - a_i = sum of h_j values of all pages pointing to it
  - h_i = sum of a_j values of all pages it points to
  - Normalise the sum of a_i values to 1
  - Normalise the sum of h_i values to 1

HITS Example

Node	B(i)	F(i)
1	2	4
2	4	134
3	2	4
4	123	2



Node	$a_0(i)$	$h_0(i)$	$a_1(i)$	$h_1(i)$	…	$a_{200}(i)$	$h_{200}(i)$
1	0.25	0.25	0.167	0.25	…	0.25	0.25
2	0.25	0.25	0.167	0.417	…	0.00	0.5
3	0.25	0.25	0.167	0.25	…	0.25	0.25
4	0.25	0.25	0.5	0.083	…	0.5	0.00

HITS vs PageRank vs LSI vs …

- Under what circumstances can we use each?
- What are the advantages/disadvantages of each?
- How do they compare to traditional boolean/vector searching?

References

- Arasu, A., J. Cho, H. Garcia-Molina, A. Paepcke and S. Raghavan (2001). “Searching the Web”, ACM Transactions on Internet Technology, Vol 1., No. 1, August 2001, pp. 2-43.
- Bell, T. C., J. G. Cleary and I. H. Witten (1990) Text Compression, Prentice Hall, New Jersey.
- Berry, M. W. and M. Browne (1999) Understanding Search Engines: Mathematical Modelling and Text Retrieval, SIAM, Philadelphia.
- Deerwester, S., S. T. Dumais, T. K. Landauer, G. W. Furnas and R. A. Harshman (1990). “Indexing by latent semantic analysis”, Journal of the Society for Information Science, Vol. 41, No. 6, pp. 391-407.
- Witten, I. H., A. Moffat and T. C. Bell (1999) Managing Gigabytes, Morgan Kauffman, San Francisco.