VU LEARNING HUB

vulearninghub.xyz

HIGHLIGHTED HANDOUT

CS726

Information Retrieval Techniques

CS726 by VU Learning HUB

Prepared for VU students • Free study resources

https://vulearninghub.xyz

CS726

Lecture # 1

- Introduction

Library Management System

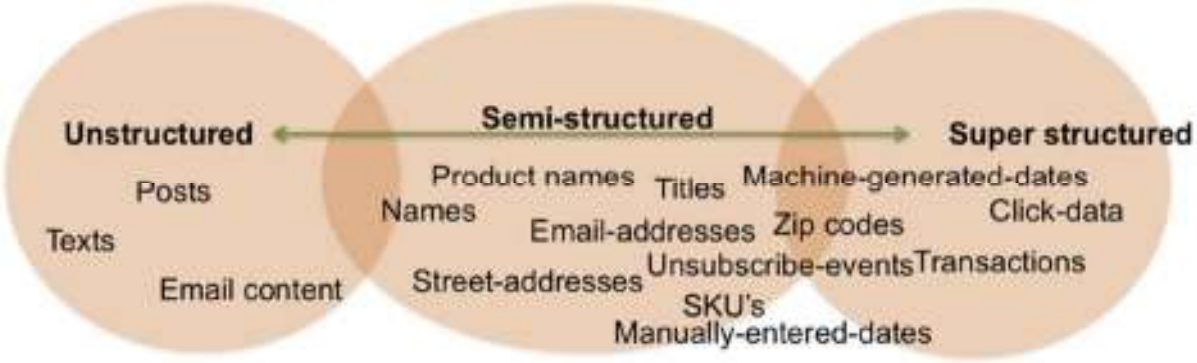
Structured Data Storage / Tables

Parameters

	Name	Native Name	Data Type	Direction	Default Value
1	ProcessID	ProcessID	String	Input	'%'
2	ProcessName	ProcessName	String	Input	'%'
3	ProcessInstanceOID	ProcessInstanceOID	String	Input	'%'
4	ProcessInstanceState	ProcessInstanceState	String	Input	'%'
5	StartDate	StartDate	String	Input	'2010-01-01 00...
6	EndDate	EndDate	String	Input	'2010-02-23 11...
7	ProcessDuration	ProcessDuration	String	Input	'%'
8	ProcessWorktime	ProcessWorktime	String	Input	'%'
9	UseIntervalEnds	UseIntervalEnds	String	Input	'true'
10	UseOuterJoins	UseOuterJoins	String	Input	'true'
11	IntervalPiStateFilter	IntervalPiStateFilter	String	Input	'started'
12	SubProcessID	SubProcessID	String	Input	'%'
13	isMemberValueFilter	isMemberValueFilter	String	Input	''
14	SupportCaseData_StartdateValueFilter	SupportCaseData_...	String	Input	'2010-02-23'

Compiled by: Farhan Ahmed **VU ID:** MS160401398

Semi-Structured and Unstructured

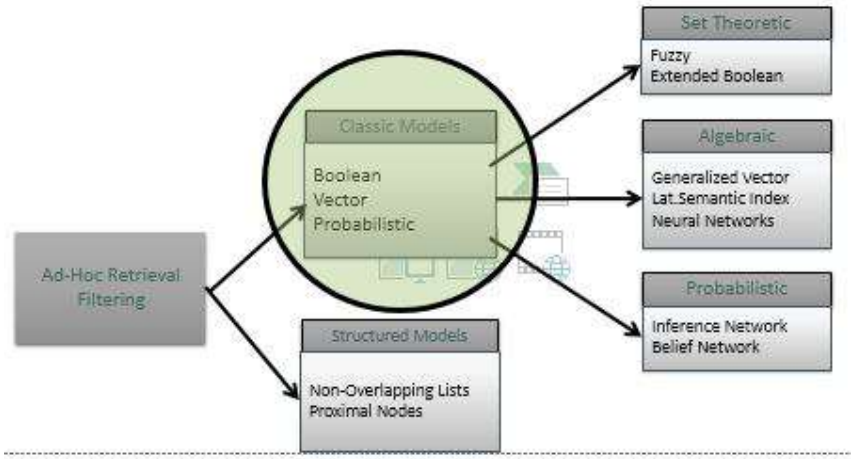


Employee Department Salary

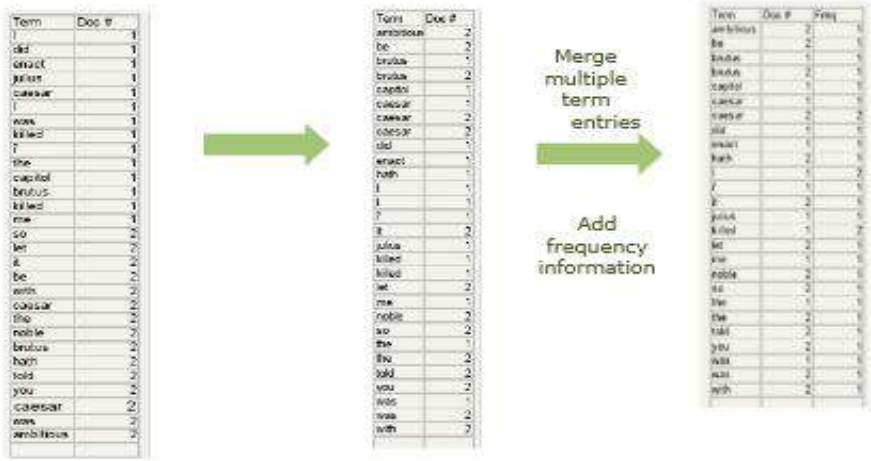
Department	Name		Salary
R & D	Fran	Whitney	$45,700.00
R & D	Matthew	Cobb	$62,000.00
R & D	Robert	Breault	$57,490.00
R & D	Natasha	Shishov	$72,995.00
R & D	Kurt	Driscoll	$48,023.69
R & D	Rodrigo	Guevara	$42,998.00
R & D	Ram	Gowda	$59,840.00
R & D	Terry	Melkisetian	$48,500.00
R & D	Lynn	Pastor	$74,500.00
R & D	Kim	Lull	$87,900.00

Library Digitization

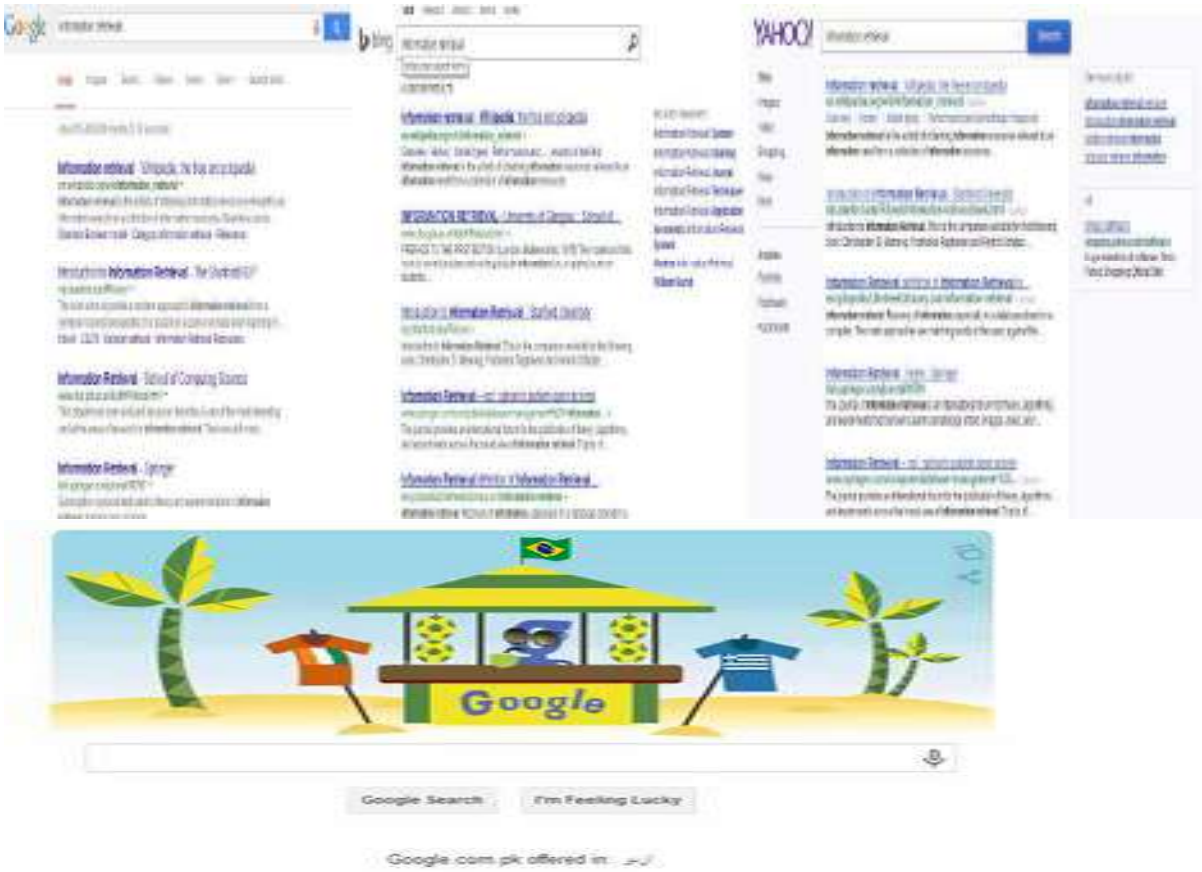
Information Retrieval Models



Indexes / Inverted Indexes

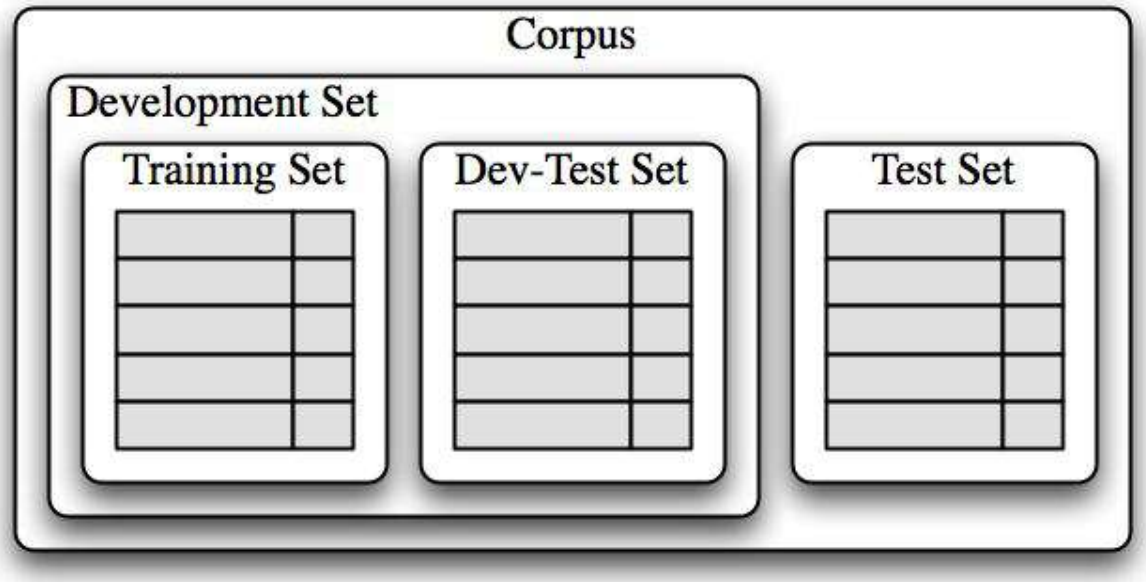


Google / Bing / Yahoo



Number of Results

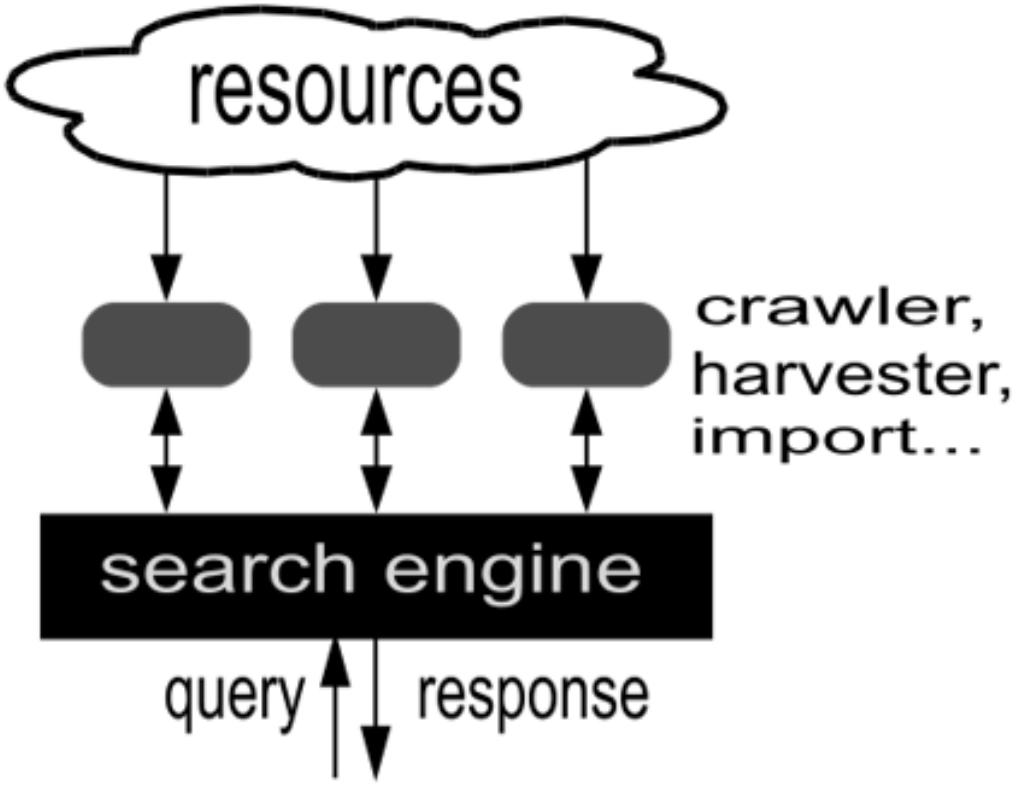
Test Corpus



The probabilistic model

The probabilistic model computes the similarity coefficient between queries and documents as the probability that a document will be relevant to a query
Search Computing
Books

Search Computing



Books

Lecture # 2

- Introduction
- Information Retrieval Models
- Boolean Retrieval Model

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- What is Information Retrieval ? ? ?
- IR Models
- The Boolean Model
- Considerations on the Boolean Model

Introduction

- What is Information Retrieval ? ? ?
- Information retrieval (IR) deals with the *representation*, *storage*, *organization of*, and *access to* **information items**.

Baeza-Yates, Ribeiro-Nieto, 1999

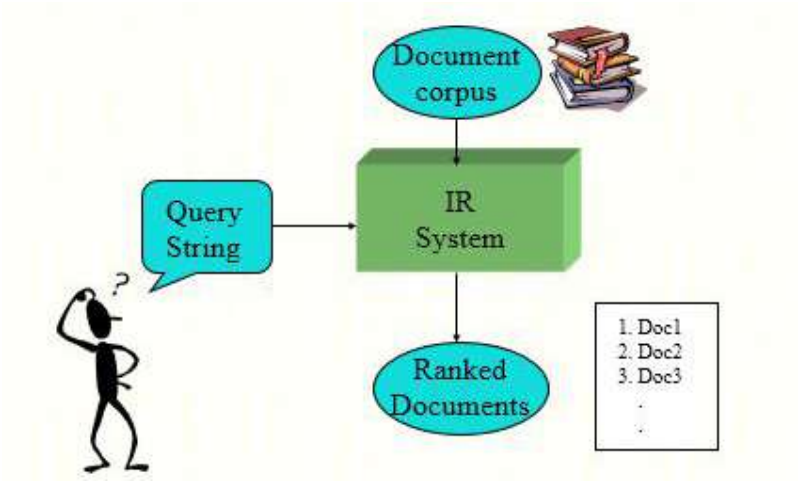
- Information retrieval (IR) is devoted to finding **relevant** documents, not finding simple match to patterns.

Grossman - Frieder, 2004

- Information retrieval (IR) is finding material (usually documents) of an **unstructured nature** (usually text) that satisfy an **information need** from within **large collections** (usually stored on computers).

Manning et al., 2007

Information Retrieval



IR Models

- **Modeling** in IR is a complex process aimed at producing a ranking function
- **Ranking function**: a function that assigns scores to documents with regard to a given query
- This process consists of two main tasks:
- The conception of a logical framework for representing documents and queries
- The definition of a ranking function that allows quantifying the similarities among documents and queries

IR Models

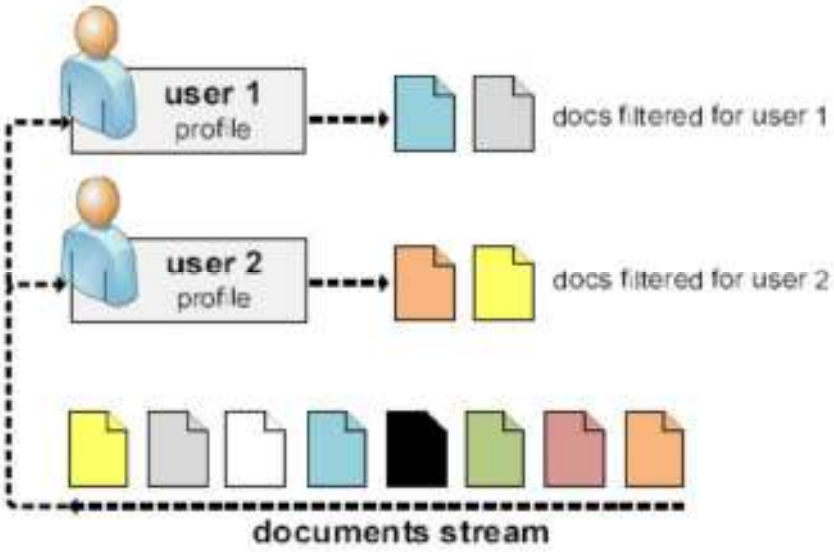
- An **IR model** is a quadruple [D, Q, F, R(qi, dj)] where
- 1. D is a set of logical views for the documents in the collection
- 2. Q is a set of logical views for the user queries
- 3. F is a framework for modeling documents and queries
- 4. R(qi, dj) is a ranking function

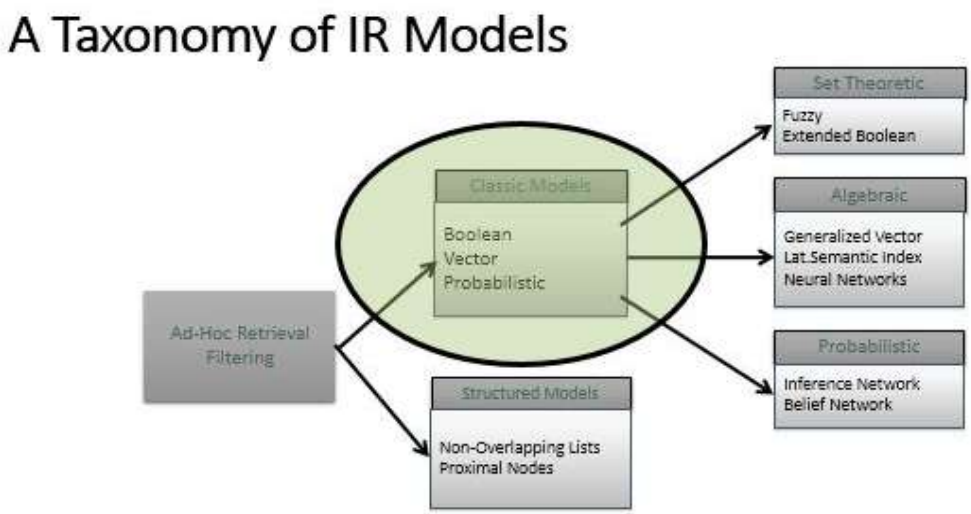
Retrieval: Ad Hoc x Filtering

- Ad Hoc Retrieval:

Retrieval: Ad Hoc x Filtering

- Filtering





The Boolean Model

- The Boolean retrieval model is a simple retrieval model based on set theory and Boolean algebra
  - Index term's Significance represented by binary weights
    - Wkj ∈{0,1} is associated to the tuple (tk, dj)
    - Rdj → set of index terms for a document
    - Rti →set of document for an index term
- Queries are defined as Boolean expressions over index terms (using Boolean operators AND, OR and NOT)
  - Brutus AND Caesar but NOT Calpurnia?

The Boolean Model (Cont…..)

- Relevance is modeled as a binary property of the documents (SC = 0 or SC=1)
  - Closed world assumption: the absence of a term t in the representation of a document d is equivalent to the presence of (not t) in the same representation

The Boolean Model (Cont…..)

$$d_1 = [1,1,1]^T$$

$$d_2 = [1,0,0]^T$$

$d_3 = [0,1,0]^T$

$R_{t1} = \{d_1, d_2\}$ $R_{t2} = \{d_1, d_3\}$ $R_{t3} = \{d_1\}$

The Boolean Model (Cont…..)

- Consider processing the query:
  - $q = t_a \wedge t_b$
    - Locate t_a in the Dictionary;
    - Retrieve its postings.
    - Locate t_b in the Dictionary;
    - Retrieve its postings.
    - "Merge" the two postings:

Posting list intersection

- The *intersection* operation is the crucial one: we need to efficiently intersect postings lists so as to be able to quickly find documents that contain both terms.
- If the list lengths

  are *x* and *y*, the

  merge takes O(*x*+*y*)

  operations.
- Crucial: postings

  sorted by docID.

Evaluation of Boolean queries

- **keyword**: retrieval of all the documents **containing** a keyword in the inverted list and build of a document list
- **OR**: creation of a list associated with the node which is the **union** of the lists associated with the left and right sub-trees
- **AND**: creation of a list associated with the node which is the **intersection** of the lists associated with the left and right sub-trees
- **BUT** = **AND NOT**: creation of a list associated with the **difference** between the lists related with the left and right sub-trees

Query optimization

- *Is the process of selecting how to organize the work of answering a query*
  - *GOAL: minimize the total amount of work performed by the system.*

Order of evaluation

$t_1 = \{d_1, d_3, d_5, d_7\}$

$t_2 = \{d_2, d_3, d_4, d_5, d_6\}$ $q = t_1$ AND t_2 OR t_3

$t_3 = \{d_4, d_6, d_8\}$

- From the left: {d3, d5} ∪ {d4, d6, d8}= {d3, d5,d4, d6, d8}
- From the right: {d2,d3,d4,d5,d6,d8} ∩ {d1, d3, d5, d7}={d3,d5}
- Standard evaluation priority: and, not, or

A and B or C and B → (A and B) or (C and B)

Considerations on the Boolean Model

- Strategy is based on a **binary decision criterion** (i.e., a document is predicted to be either relevant or non-relevant) without any notion of a grading scale
- The Boolean model is in reality much more a data (instead of information) retrieval model

Pros:

- Boolean expressions have precise semantics
- Structured queries
- For expert users, intuitivity
- Simple and neat formalism → great attention in past years and was adopted by many of the early commercial bibliographic systems

Drawbacks of the Boolean Model

- Retrieval based on binary decision criteria with no notion of partial matching
- No ranking of the documents is provided (absence of a grading scale)
- Information need has to be translated into a Boolean expression, which most users find awkward
- The Boolean queries formulated by the users are most often too simplistic
- The model frequently returns either too few or too many documents in response to a user query

Resources

- Modern Information Retrieval
- Chapter 1 of IIR
- Resources at http://ifnlp.org/ir
  - Boolean Retrieval

Lecture # 3

- Boolean Retrieval Model
- Rank Retrieval Model

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Boolean Retrieval Model
- Information Retrieval Ingredients
- Westlaw
- Ranked retrieval models

Boolean Retrieval Model

D_1 = {This is a pen} D_2 = {It is a pen}
Set (D_1, D_2) = {This, It, is, a, pen}
Set = {a, b, c} = {b, a, c} bag = {a, a, b, c}

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join
  - query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
  - You know exactly what you are getting.

- Many search systems you use are also Boolean: spotlight, email, intranet etc.

Information Retrieval Ingredients

- Documents representation
- Query formulation
- Query processing

Westlaw

Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.
- In 2005, Boolean search (called "Terms and Connectors" by Westlaw) was still the default, and used by a large percentage of users . . .
- . . . although ranked retrieval has been available since 1992.

Westlaw: Example queries

- *Information need*: Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company
- *Query*: "trade secret" /s disclos! /s prevent /s employe!
- *Information need*: Requirements for disabled people to be able to access a workplace
- Query: disab! /p access! /s work-site work-place (employment /3 place)
- *Information need*: Cases about a host's responsibility for drunk guests
- *Query*: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Problem with Boolean search: feast or famine

- Requires query writing skills

- Boolean queries often result in either too few (=0) or too many (1000s) 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

- Rather than a set of documents satisfying a query expression, in **ranked retrieval**, the system returns an ordering over the (top) documents in the collection for a query
- **Free text queries**: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here, but 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
    - Indeed, the size of the result set is 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

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match".

• Resources

- Chapter 1 of IIR
- Resources at http://ifnlp.org/ir
    - Boolean Retrieval

Lecture # 4

- Vector Space Retrieval Model

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- The Vector Model
- Term frequency tf
- Document frequency
- tf-idf weighting

The Vector Model (Salton, 1968)

- The *Vector Space Model* represents documents and queries as vectors in the term space
    - Term weights are used to compute the ***degree of similarity (or score) between each document stored in the system and the user query***
        - By sorting in decreasing order of this degree of similarity, the vector model takes into consideration documents which **match the query terms only partially**.
        - Ranked document answer set is a lot more **precise** (in the sense that it better matches the user information need) than the document answer set retrieved by the Boolean model
- A measure of the **similarity** between the two vectors is then computed
    - Documents whose content (terms in the document) correspond most closely to the content of the query are judged to be the most relevant

• Binary term-document incidence matrix

	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

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

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document 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 document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.
- In a sense, this is a step back: The positional index is required to distinguish these two documents.
- We will look at "recovering" positional information later in this course.
- For now: bag of words model

Term frequency tf

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

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

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

Term Frequency Normalization

- We can use another normalization method for tf.
- Find the term frequencies of all terms in a document.
- Find out the maximum tf_{max} from all those terms.
- The tf of term '*i*' referred to as tf_i is obtained by dividing the tf_i by tf_{max}.

$$tf_i = tf_i / tf_{max}$$

- This brings the term frequencies of each term in a document between 1 and 0, irrespective of the size of a document.

Document frequency

- Rare terms are more informative than frequent terms
  - Stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Document frequency, continued

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.

Document frequency, continued

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
- → For frequent terms, we want high positive weights for words like *high, increase, and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $df_t \leq N$
- We define the idf (inverse document frequency) of t by

$$idf_t = \log_{10}(N/df_t)$$

  - We use $\log(N/df_t)$ instead of N/df_t to "dampen" the effect of idf.

Will turn out the base of the log is immaterial.

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

$$idf_t = \log_{10}(N/df_t)$$

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

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.
  $$w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / df_t)$$
- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Resources

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

Lecture # 5

- TF-IDF Weighting
- Document Representation in Vector Space
- Query Representation in Vector Space
- Similarity Measures

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Computing TF-IDF
- Mapping to vectors
- Coefficients
- Query Vector
- Similarity Measure
- Jaccard coefficient
- Inner Product

Computing TF-IDF-- An Example

- Given a document containing terms with given frequencies:
- A(3), B(2), C(1)
- Assume collection contains 10,000 documents and
- document frequencies of these terms are:
- A(50), B(1300), C(250)
- Then:
- A: tf = 3/3; idf = $\log_2$(10000/50) = 7.6; tf-idf = 7.6
- B: tf = 2/3; idf = $\log_2$ (10000/1300) = 2.9; tf-idf = 2.0
- C: tf = 1/3; idf = $\log_2$ (10000/250) = 5.3; tf-idf = 1.8

Binary → count → weight matrix

	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

	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

	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 $\in \mathbb{R}^{|V|}$

Mapping to vectors

- terms: an axis for every term
  - -vectors corresponding to terms are canonical vectors
- documents: sum of the vectors corresponding to terms in the doc
- queries: treated the same as documents

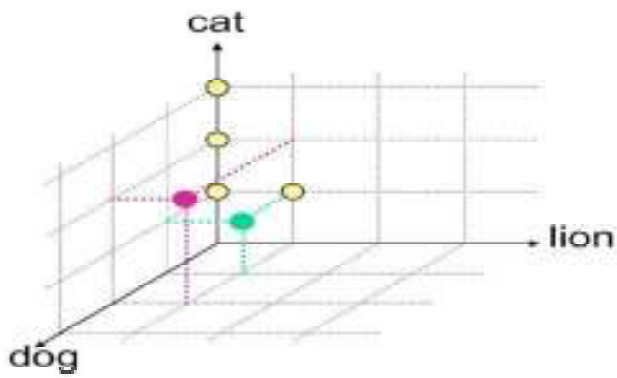
$$\begin{pmatrix} & T_1 & T_2 & \cdots & T_t \\ D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\ D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \cdots & w_{tn} \end{pmatrix}$$

Coefficients

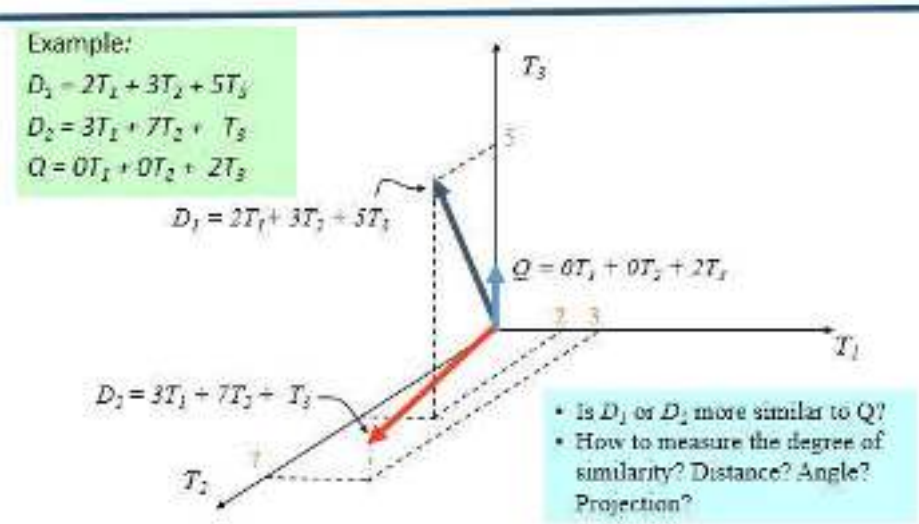
- The coefficients (vector lengths, term weights) represent term presence, importance, or "aboutness"
  - –Magnitude along each dimension
- Model gives no guidance on how to set term weights
- Some common choices:
  - –Binary: 1 = term is present, 0 = term not present in document
  - –tf: The frequency of the term in the document
  - –tf• idf: idfindicates the discriminatory power of the term
- •Tf·idfis far and away the most common
  - –Numerous variations…

Graphic Representation

- cat
- cat cat
- cat cat cat
- cat lion
- lion cat
- cat lion dog
- cat cat lion dog dog



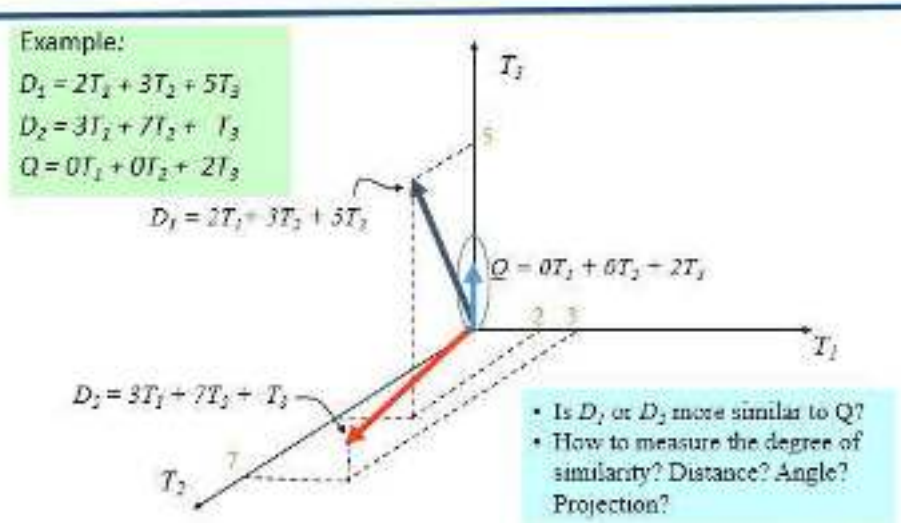
Graphic Representation (Cont)



Query Vector

- Query vector is typically treated as a document and also tf-idf weighted.
- Alternative is for the user to supply weights for the given query terms.

Graphic Representation Query Representation(Cont)



Similarity Measure

- A similarity measure is a function that computes the *degree of similarity* between two vectors.
- Using a similarity measure between the query and each document:
  - It is possible to rank the retrieved documents in the order of presumed relevance.
  - It is possible to enforce a certain threshold so that the size of the retrieved set can be controlled.

Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*
- jaccard*(A,B)* = $|A \cap B| / |A \cup B|$
- jaccard*(A,A)* = 1
- jaccard*(A,B)* = 0 if $A \cap B = 0$
- *A* and *B* don't have to be the same size.
- Always assigns a number between 0 and 1.

Issues with Jaccard for scoring

- It **doesn't consider** ***term frequency*** (how many times a term occurs in a document)
- **Rare terms in a collection are more informative** than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length
- . . . instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization.

$$|A \cap B| / \sqrt{|A \cup B|}$$

Similarity Measure- Inner Product

- Similarity between vectors for the document $\boldsymbol{d}_i$ and query $\boldsymbol{q}$ can be computed as the vector inner product (a.k.a. dot product):

$\text{sim}(\boldsymbol{d}_j, \boldsymbol{q}) = \boldsymbol{d}_j \bullet \boldsymbol{q} =$

where wij is the weight of term i in document j and wiq is the weight of term i in the query

- For binary vectors, the inner product is the number of matched query terms in the document (size of intersection).
- For weighted term vectors, it is the sum of the products of the weights of the matched terms.

Inner Product -- Examples

Binary: retrieval database architecture computer text management information

- D = 1, 1, 1, 0, 1, 1, 0
- Q = 1, 0, 1, 0, 0, 1, 1

Size of vector = size of vocabulary = 7
0 means corresponding term not found in document or query

sim(D, Q) = 3

Weighted:

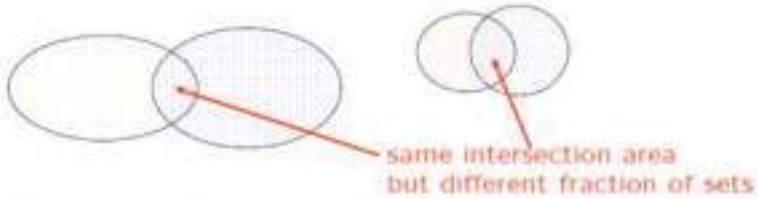
$D_1 = 2T_1 + 3T_2 + 5T_3 \qquad D_2 = 3T_1 + 7T_2 + 1T_3$

$Q = 0T_1 + 0T_2 + 2T_3$

$\text{sim}(D_1, Q) = 2*0 + 3*0 + 5*2 = 10$

$\text{sim}(D_2, Q) = 3*0 + 7*0 + 1*2 = 2$

Similarity, Normalized



$$similarity = \frac{|\text{intersection}|}{|\text{set}_1| \cdot |\text{set}_2|}$$

- the size of intersection alone is meaningless
- often divided by sizes of sets
- same for vectors, using norm
  - –by normalizing vectors, cosine does not change

Resources

- Many slides in this section are adapted from Prof. Joydeep Ghosh (UT ECE) who in turn adapted them from Prof. Dik Lee (Univ. of Science and Tech, Hong Kong)

Lecture # 6

- Similarity Measures
- Cosine Similarity Measure

ACKNOWLEDGEMENTS

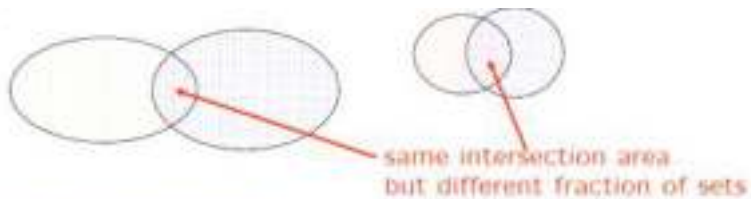
The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Cosine Similarity
- Basic indexing pipeline
- Sparse Vectors
- Inverted Index

Similarity, Normalized



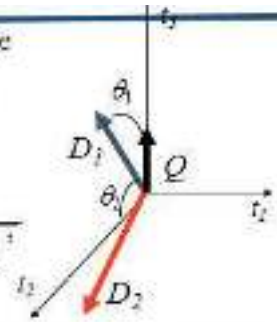
$$similarity = \frac{|\text{intersection}|}{|\text{set}_1| \cdot |\text{set}_2|}$$

- the size of intersection alone is meaningless
- often divided by sizes of sets
- same for vectors, using norm
  - –by normalizing vectors, cosine does not change

Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\mathrm{CosSim}(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{iq}^2}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ $\quad \mathrm{CosSim}(D_1, Q) = 10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$

$D_2 = 3T_1 + 7T_2 + 1T_3$ $\quad \mathrm{CosSim}(D_2, Q) = 2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$

$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

Common similarity measures

Sim(X,Y)	Binary Term Vectors	Weighted Term Vectors
Inner product	$\lvert X \cap Y \rvert$	$\sum x_i \cdot y_i$
Dice coefficient	$\frac{2\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$	$\frac{2\sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2}$
Cosine coefficient	$\frac{\lvert X \cap Y \rvert}{\sqrt{\lvert X \rvert}\sqrt{\lvert Y \rvert}}$	$\frac{\sum x_i \cdot y_i}{\sqrt{\sum x_i^2 \cdot \sum y_i^2}}$
Jaccard coefficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert - \lvert X \cap Y \rvert}$	$\frac{\sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2 - \sum x_i \cdot y_i}$

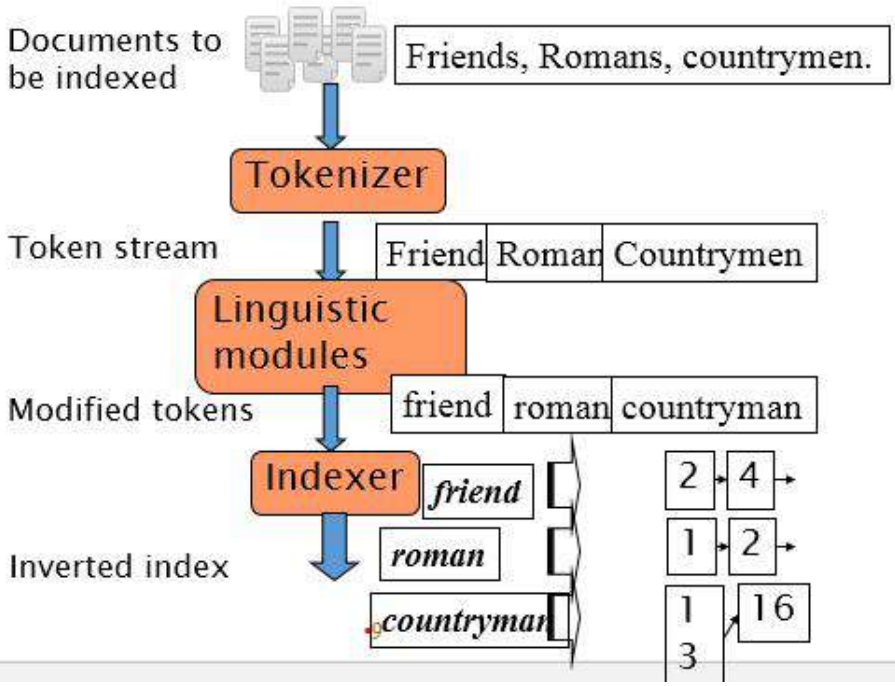
Vector space retrieval: summary

- Standard vector space
  - –Each dimension corresponds to a term in the vocabulary
  - –Vector elements are real-valued, reflecting term importance
  - –Any vector (document, query, ...) can be compared to any other
  - –Cosine correlation is the similarity metric used most often
    - Provides partial matching and ranked results.

Vector space model: Disadvantages

- Assumed independence relationship among terms
  - –Though this is a very common retrieval model assumption
- Lack of justification for some vector operations
  - –e.g. choice of similarity function
  - –e.g., choice of term weights
- Barely a retrieval model
  - –Doesn't explicitly model relevance, a person's information need, language models, etc.
- Assumes a query and a document can be treated the same (symmetric)

Basic indexing pipeline



Sparse Vectors

- Vocabulary and therefore dimensionality of vectors can be very large, ~10^4 .
- However, most documents and queries do not contain most words, so vectors are sparse (i.e. most entries are 0).
- Need efficient methods for storing and computing with sparse vectors.

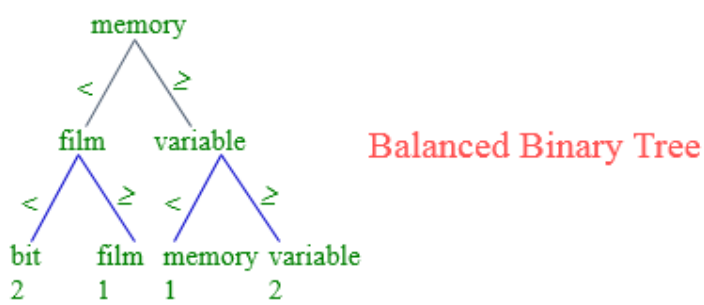
Sparse Vectors as Lists

- Store vectors as linked lists of non-zero-weight tokens paired with a weight.
    - Space proportional to number of unique tokens (*n*) in document.
    - Requires linear search of the list to find (or change) the weight of a specific token.
    - Requires quadratic time in worst case to compute vector for a document:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

Sparse Vectors as Trees

- Index tokens in a document in a balanced binary tree or trie with weights stored with tokens at the leaves.



Sparse Vectors as Trees (cont.)

- Space overhead for tree structure: ~2*n* nodes.
- O(log *n*) time to find or update weight of a specific token.
- O(*n* log *n*) time to construct vector.
- Need software package to support such data structures.

Sparse Vectors as HashTables

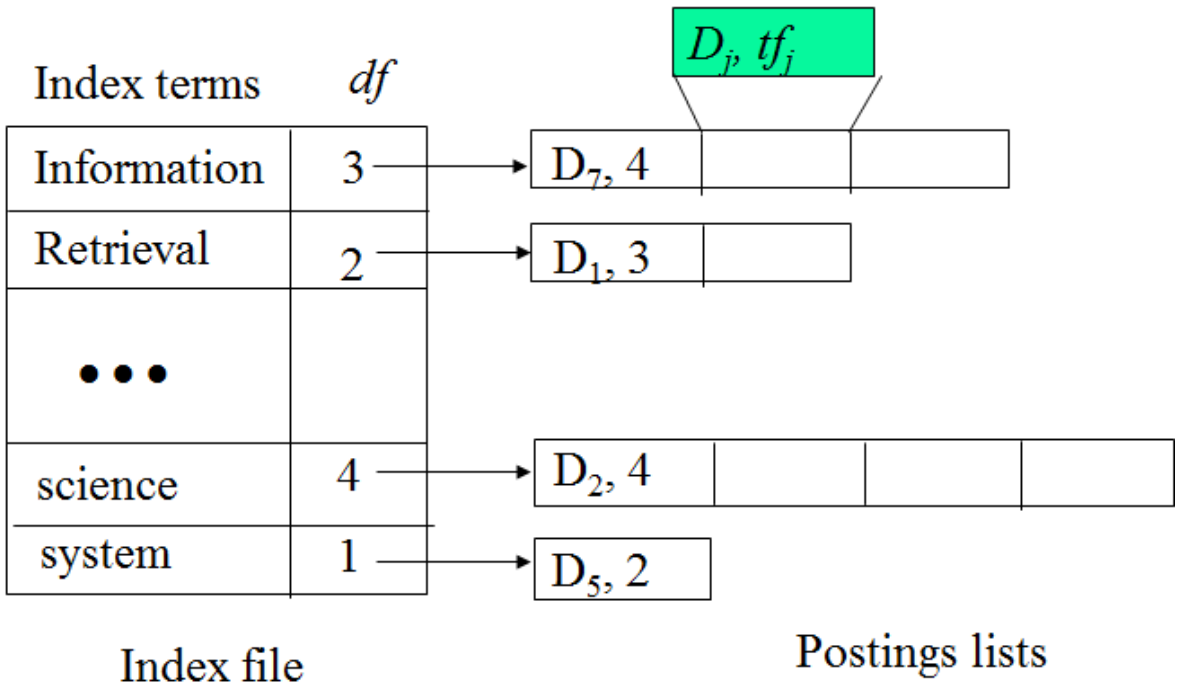
- Store tokens in hashtable, with token string as key and weight as value.
- Storage overhead for hashtable ~1.5*n.*
- Table must fit in main memory.
- Constant time to find or update weight of a specific token (ignoring collisions).

- O(n) time to construct vector (ignoring collisions).

Implementation Based on Inverted Files

- In practice, document vectors are not stored directly; an inverted organization provides much better efficiency.
- The keyword-to-document index can be implemented as a hash table, a sorted array, or a tree-based data structure (trie, B-tree).
- Critical issue is logarithmic or constant-time access to token information.

Inverted Index



Lecture # 7

- Parsing Documents

ACKNOWLEDGEMENTS

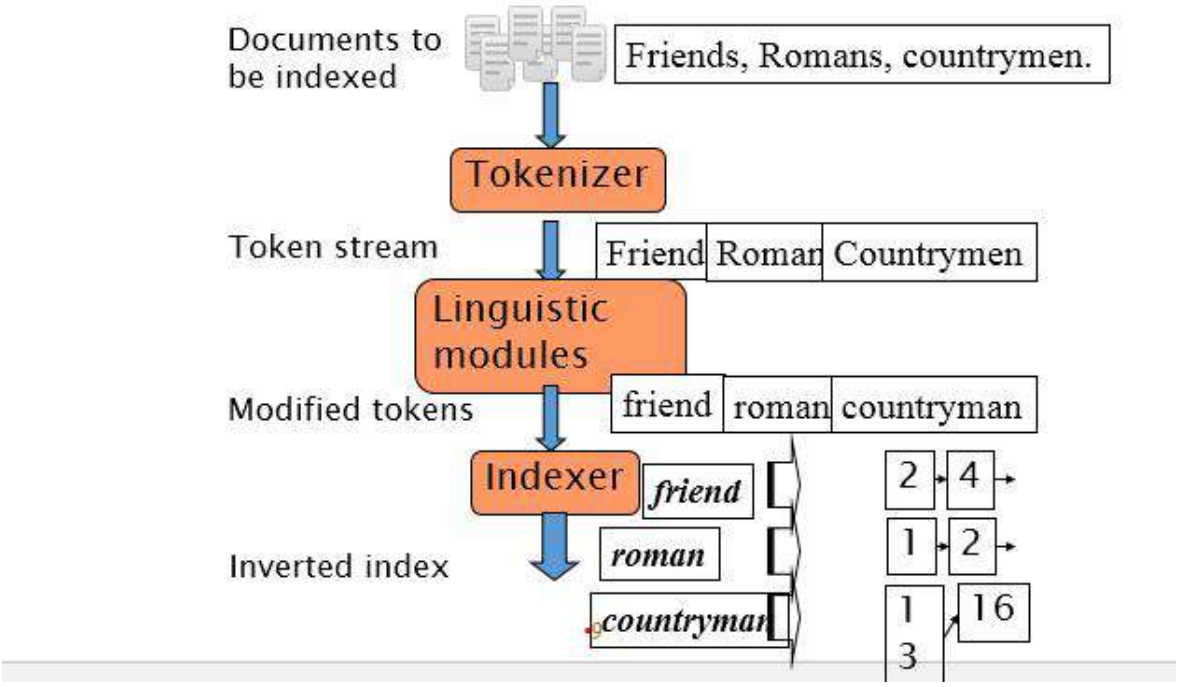
The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Basic indexing pipeline
- Inverted Index
- Cosine Similarity Measure
- Time Complexity of Indexing
- Retrieval with an Inverted Index
- Inverted Query Retrieval Efficiency

Basic indexing pipeline



Sparse Vectors

- Vocabulary and therefore dimensionality of vectors can be very large, ~10^4.
- However, most documents and queries do not contain most words, so vectors are sparse (i.e. most entries are 0).
- Need efficient methods for storing and computing with sparse vectors.

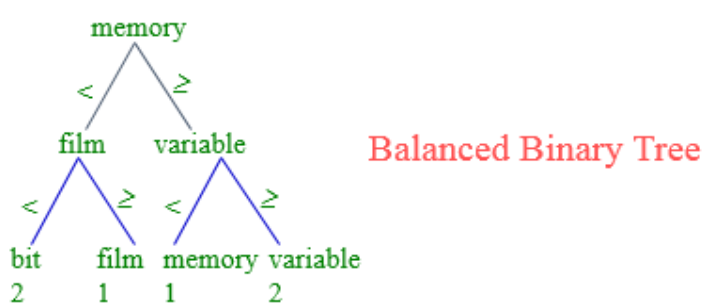
Sparse Vectors as Lists

- Store vectors as linked lists of non-zero-weight tokens paired with a weight.
  - Space proportional to number of unique tokens (*n*) in document.
  - Requires linear search of the list to find (or change) the weight of a specific token.
  - Requires quadratic time in worst case to compute vector for a document:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

Sparse Vectors as Trees

- Index tokens in a document in a balanced binary tree or trie with weights stored with tokens at the leaves.



Sparse Vectors as Trees (cont.)

- Space overhead for tree structure: ~2*n* nodes.
- O(log *n*) time to find or update weight of a specific token.
- O(*n* log *n*) time to construct vector.
- Need software package to support such data structures.

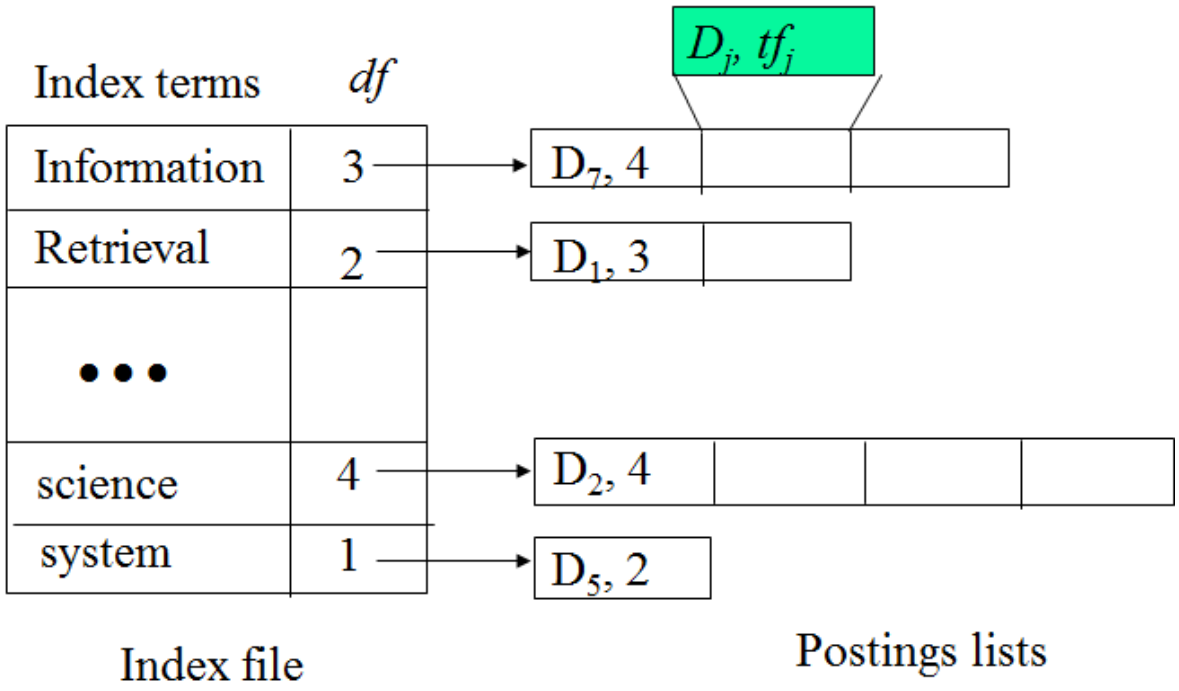
Sparse Vectors as HashTables

- Store tokens in hashtable, with token string as key and weight as value.
- Storage overhead for hashtable ~1.5*n.*
- Table must fit in main memory.
- Constant time to find or update weight of a specific token (ignoring collisions).
- O(*n*) time to construct vector (ignoring collisions).

Implementation Based on Inverted Files

- In practice, document vectors are not stored directly; an inverted organization provides much better efficiency.
- The keyword-to-document index can be implemented as a hash table, a sorted array, or a tree-based data structure (trie, B-tree).
- Critical issue is logarithmic or constant-time access to token information.

Inverted Index



Creating an Inverted Index

```
Create an empty HashMap, H;
For each document, D, (i.e. file in an input directory):
    Create a HashMapVector,V, for D;
      For each (non-zero) token, T, in V:
            If T is not already in H, create an empty
             TokenInfo for T and insert it into H;
          Create a TokenOccurence for T in D and
               add it to the occList in the TokenInfo for T;
Compute IDF for all tokens in H;
Compute vector lengths for all documents in H;
```

Computing IDF

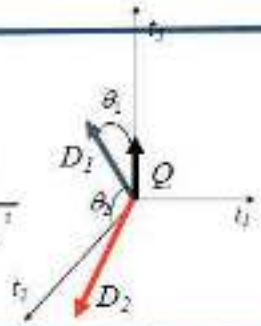
```
Let N be the total number of Documents;
For each token, T, in H:
    Determine the total number of documents, M,
      in which T occurs (the length of T's occList);
    Set the IDF for T to log(N/M);
```

Note this requires a second pass through all the tokens after all documents have been indexed.

Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\mathrm{CosSim}(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j|\,|\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



$D_1 = 3T_1 + 3T_2 + 5T_3$ $\quad \mathrm{CosSim}(D_1, Q) = 10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$

$D_2 = 3T_1 + 7T_2 + 1T_3$ $\quad \mathrm{CosSim}(D_2, Q) = 2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$

$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

Document Vector Length

- Remember that the length of a document vector is the square-root of sum of the squares of the weights of its tokens.
- Remember the weight of a token is:

TF * IDF

- Therefore, must wait until IDF's are known (and therefore until all documents are indexed) before document lengths can be determined.

Computing Document Lengths

Assume the length of all document vectors (stored in the DocumentReference) are initialized to 0.0;

For each token T in H:

Let, I, be the IDF weight of T;

For each TokenOccurence of T in document D

Let, C, be the count of T in D;

Increment the length of D by $(I*C)^2$;

For each document D in H:

Set the length of D to be the square-root of the

current stored length;

Time Complexity of Indexing

- Complexity of creating vector and indexing a document of *n* tokens is O(*n*).
- So indexing *m* such documents is O(*m n*).
- Computing token IDFs for a vocabularly *V* is O(|*V*|).
- Computing vector lengths is also O(*m n*).
- Since |*V*| ≤ *m n,* complete process is O(*m n*), which is also the complexity of just reading in the corpus.

Retrieval with an Inverted Index

- Tokens that are not in both the query and the document do not effect cosine similarity.
    - Product of token weights is zero and does not contribute to the dot product.
- Usually the query is fairly short, and therefore its vector is *extremely* sparse.
- Use inverted index to find the limited set of documents that contain at least one of the query words.

Inverted Query Retrieval Efficiency

- Assume that, on average, a query word appears in *B* documents:

- Then retrieval time is O(|*Q*| *B*), which is typically, much better than naïve retrieval that examines all *N* documents, O(|*V*| *N*), because |*Q*| << |*V*| and *B* << *N*.

User Interface

Until user terminates with an empty query:

Prompt user to type a query, Q.

Compute the ranked array of retrievals R for Q;

Print the name of top N documents in R;

Until user terminates with an empty command:

Prompt user for a command for this query result:

1) Show next N retrievals;

2) Show the Mth retrieved document;

Lecture # 8

- Token
- Numbers
- Stop Words

ACKNOWLEDGEMENTS

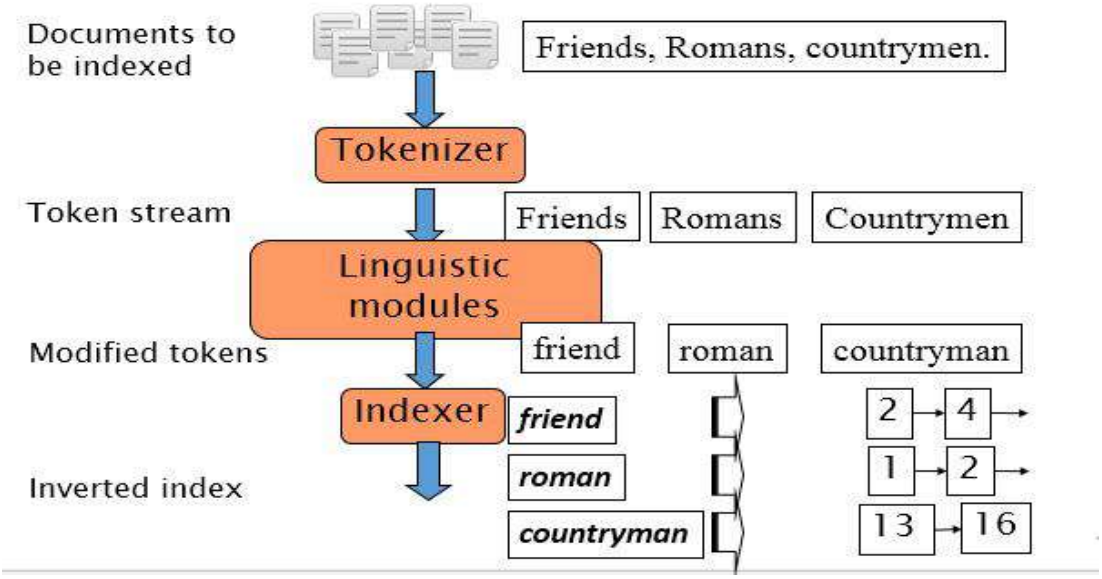
The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Parsing a document
- Complications: Format/language
- Precision and Recall
- Tokenization
- Numbers
- Tokenization: language issues
- Stop words

Basic indexing pipeline



Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
  - (CP1252, UTF-8, …)

These tasks are often done heuristically …

Complications: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may contain terms from many languages.
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
  - French email quote clauses from an English-language contract
- There are commercial and open source libraries that can handle a lot of this stuff

Complications: What is a document?

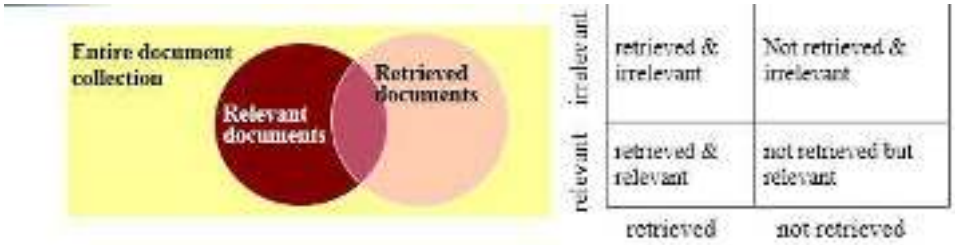
We return from our query "documents" but there are often interesting questions of grain size:

What is a unit document?

- A file?
- An email? (Perhaps one of many in a single mbox file)
  - What about an email with 5 attachments?

- A group of files (e.g., PPT or LaTeX split over HTML pages)

Precision and Recall



$$recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents}$$

$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- A token is an instance of a sequence of characters
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** →

Finland AND ***s***? ***Finlands***? ***Finland's***?

  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***state-of-the-art***: break up hyphenated sequence.
    - ***co-education***
    - ***lowercase***, ***lower-case***, ***lower case*** ?
    - It can be effective to get the user to put in possible hyphens
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

Numbers

- ***3/20/91*** ***Mar. 12, 1991*** ***20/3/91***
- ***55 B.C.***
- ***B-52***
- ***My PGP key is 324a3df234cb23e***
- ***(800) 234-2333***
    - Often have embedded spaces
    - Older IR systems may not index numbers
        - But often very useful: think about things like looking up error codes/stacktraces on the web
        - (One answer is using n-grams: IIR ch. 3)
    - Will often index "meta-data" separately
        - Creation date, format, etc.

Tokenization: language issues

- French
    - ***L'ensemble*** → one token or two?
        - ***L*** ? ***L'*** ? ***Le*** ?
        - Want ***l'ensemble*** to match with ***un ensemble***
            - Until at least 2003, it didn't on Google
                - Internationalization!
- German noun compounds are not segmented
    - ***Lebensversicherungsgesellschaftsangestellter***
    - 'life insurance company employee'
    - German retrieval systems benefit greatly from a **compound splitter** module
        - Can give a 15% performance boost for German

Tokenization: language issues

- Chinese and Japanese have no spaces between words

莎拉波娃现在居住在美国东南部的佛罗里达。

- Not always guaranteed and unique tokenization
- Further complication in Japanese, with multiple alphabets intermingled
- Dates/amount in multiple format

Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures
- استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.
- 'Algeria achieved its independence in 1962 after 132 years of French occupation.'
- With Unicode, the surface presentation is complex, but the stored form is straightforward

Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
    - They have little semantic content: *the, a, and, to, be*
    - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
    - Good compression techniques (IIR 5) means the space for including stop words in a system is very small
    - Good query optimization techniques (IIR 7) mean you pay little at query time for including stop words.
    - You need them for:
        - Phrase queries: "King of Denmark"
        - Various song titles, etc.: "Let it be", "To be or not to be"
        - "Relational" queries: "flights to London"

Resources

- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer: http//www.sims.berkeley.edu/~hearst/irbook/porter.html
- H.E. Williams, J. Zobel, and D. Bahle, "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.

http://www.seg.rmit.edu.au/research/research.php?author=4

Lecture # 9

- Terms Normalization

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

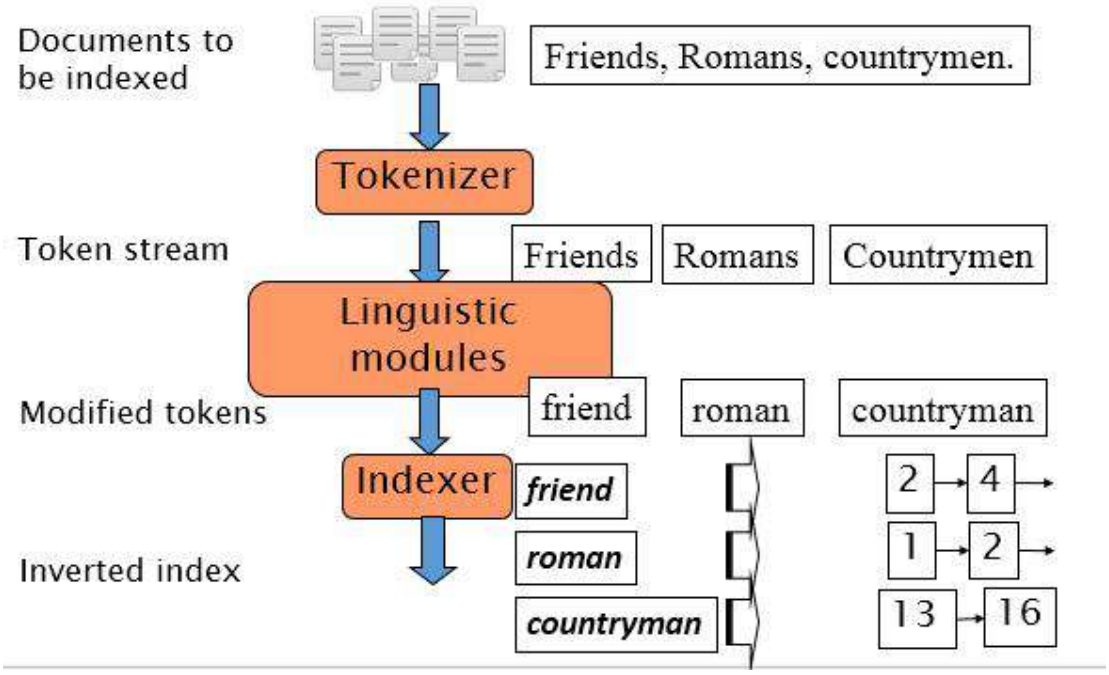
1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Normalization
- Case folding
- Normalization to terms
- Thesauri and soundex

Compiled by: Farhan Ahmed **VU ID:** MS160401398

Basic indexing pipeline



Normalization to terms

- We may need to "normalize" words in indexed text as well as query words into the same form
    - We want to match ***U.S.A.*** and ***USA***
- **Tokens are transformed to terms which are then entered into the index**
- A term is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly define equivalence classes of terms by, e.g.,
    - deleting periods to form a term
        - ***U.S.A., USA*** \ ***USA***
    - deleting hyphens to form a term
        - ***anti-discriminatory, antidiscriminatory*** \ ***antidiscriminatory***

1. Normalization: other languages

- Accents: e.g., French ***résumé*** vs. ***resume*.**
    - **Simple remedy remove accent but not good in case of Resume with and without accent.**

- **Cliché = Cliche (with and without accent same meaning)**

- **Important consideration:** Are the users going to use accents while writing queries?
- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
  - Should be equivalent
  - Even in languages that standard have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - ***Tuebingen, Tübingen, Tubingen \ Tubingen***

Normalization: other languages

- Normalization of things like date forms
  - ***7 月 30 日 vs. 7/30 (date or mathematical expression) → Diversification: 7/30 = 7/30, July 30, 7-30***
  - ***Japanese use of kana vs. Chinese characters***
    - ***In Japanese there are several different character sets and normalization needs to take care of this fact, and it should be able to resolve the query entered using any char-set***
  - ***In German MIT is a word, so how to differentiate the usage if it is University of the word MIT?***
- **Tokenization** and **normalization** may depend on the language and so is intertwined with language detection
- Same method of normalization should be used while indexing as well as while query processing

2. Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail

- Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…
- A word starting with a capital letter in the middle of sentence is for nouns, so case folding may be given importance in this case.
- However, if users are not going to use capital letters then there is no point in improving index.
- Longstanding Google example: [fixed in 2011…]
- Query C.A.T.
- #1 result is for "cats" not Caterpillar Inc.

3. Normalization to terms

- An alternative to equivalence classing is to do **asymmetric expansion**
- An example of where this may be useful
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient
  - Increases the size of the postings list, but give more control in query processing.

4. Thesauri and soundex

- Do we handle synonyms and homonyms?
- Synonym: Diff words same meanings. (Automobile / Car)
- Homonyms: Same words different meanings (Jaguar) (Blackberry)
  - For homonyms postings for all variants against same index word (term).
- For Synonym
  - E.g., by hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***
  - We can rewrite to form equivalence-class terms

        - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
    - Or we can expand a query
        - When the query contains ***automobile***, look under ***car*** as well
- What about spelling mistakes? Chebichev
    - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics
    - Groups the words that sound similar into same equivalence class.

Resources

- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer: http//www.sims.berkeley.edu/~hearst/irbook/porter.html
- H.E. Williams, J. Zobel, and D. Bahle, "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.

http://www.seg.rmit.edu.au/research/research.php?author=4

Lecture 10

- Lemmatization
- Stemming

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Lemmatization
- Stemming
- Porter's algorithm
- Language-specificity

5. Lemmatization

- NPL tool. It uses dictionaries and morphological analysis of words in order to return the base or dictionary form of a word
- Reduce inflectional/variant forms to base form
- E.g.,
    - *am, are, is → be*
    - *car, cars, car's, cars' → car*
- *the boy's cars are different colors → the boy car be different color*
- *No change in proper nouns e.g. Pakistan remains same*
- Lemmatization implies doing "proper" reduction to dictionary headword form
    - Example: Lemmatization of "saw" → attempts to return "see" or "saw" depending on whether the use of the token is a verb or a noun

6. Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggests crude affix chopping
    - language dependent
    - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.
    - e.g., **computation, computing, computer,** all reduce to **comput.**

for example compressed and compression are both accepted as equivalent to compress. → for exampl compress and compress ar both accept as equival to compress

Porter's algorithm

- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Typical rules in Porter

- *sses → ss* — *Processes → Process*
- *ies → I* — *Skies → Ski;* ponies → poni
- *ational → ate* — *Rotational → Rotate*
- *tional → tion* — *national → nation*
- *S → ""* — *cats → cat*
- Weight of word sensitive rules
- *(m>1) EMENT →*
- (whatever comes before emenet has length greater than 1, replace emenet with null)
  - *replacement → replac*
  - *cement → cement*

Other stemmers

- Other stemmers exist:
  - Lovins stemmer
    - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
    - Single-pass, longest suffix removal (about 250 rules)
  - Paice/Husk stemmer
  - Snowball
- Full morphological analysis (lemmatization)
  - At most modest benefits for retrieval

Stemming Example

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Language-specificity

- The above methods embody transformations that are
    - Language-specific, and often
    - Application-specific
- These are "plug-in" addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

Does stemming/lemmatization help?

- English: very mixed results. Helps recall for some queries but harms precision on others
    - E.g., operative (dentistry) ⇒ oper
    - Operational (research) => oper
    - Operating (systems) => oper
    - Increase recall but reduce precision, such normalization is not very useful in English language.
- Definitely useful for Spanish, German, Finnish, …
    - 30% performance gains for Finnish!
    - **Reason** is that there are **very clear morphological rules** so as to form words in these languages.
    - Domain specific normalization may also be helpful e.g. normalizing the words w.r.t their usage in a particular domain.

Resources

- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer: http//www.sims.berkeley.edu/~hearst/irbook/porter.html
- H.E. Williams, J. Zobel, and D. Bahle, "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.

http://www.seg.rmit.edu.au/research/research.php?author=4

Lecture # 11

- Compression

ACKNOWLEDGEMENTS

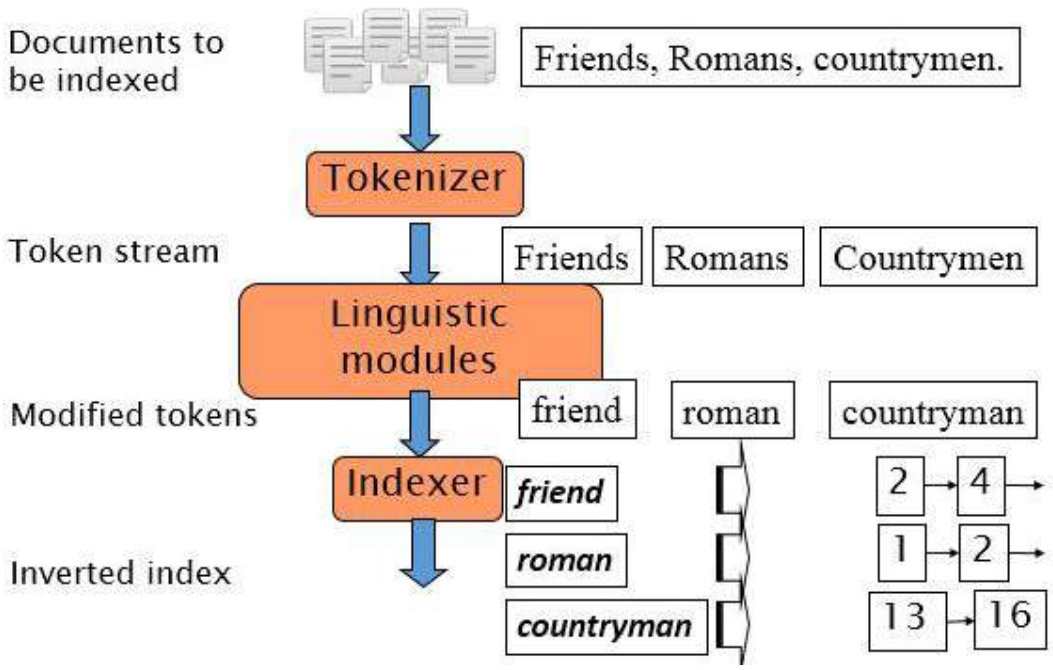
The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- compression for inverted indexes
- Dictionary storage
- Dictionary-as-a-String
- Blocking

Basic indexing pipeline

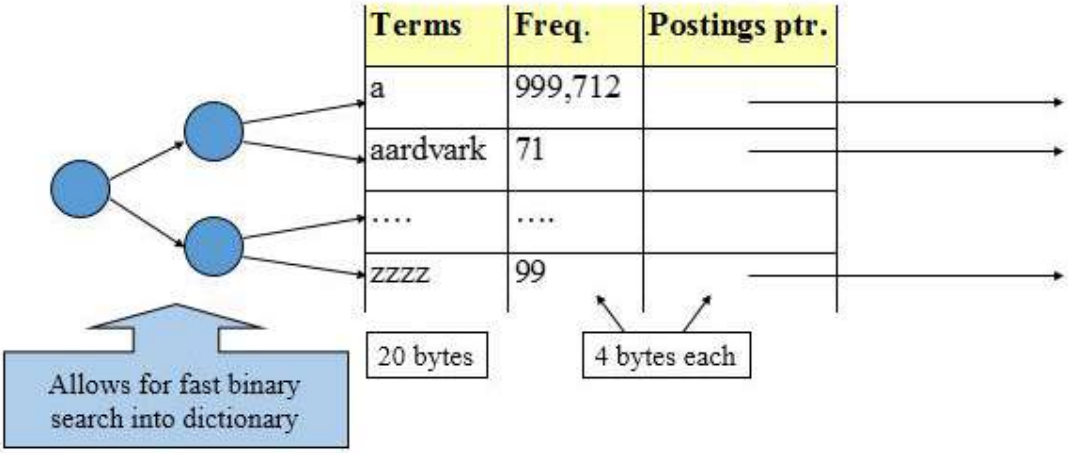


Why compression for inverted indexes?

- Dictionary
  - Make it small enough to keep in main memory
  - Make it so small that you can keep some postings lists in main memory too
- Postings file(s)
  - Reduce disk space needed
  - Decrease time needed to read postings lists from disk
  - Large search engines keep a significant part of the postings in memory.
    - Compression lets you keep more in memory

Dictionary storage- first cut

- Array of fixed-width entries
  - 500,000 terms; 28 bytes/term = 14MB.



Fixed-width terms are wasteful

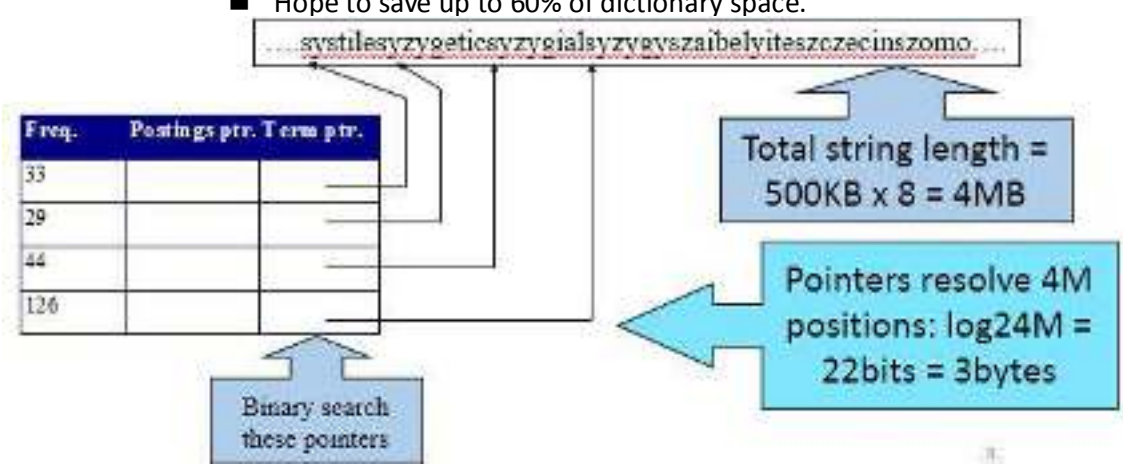
- Most of the bytes in the **Term** column are wasted – we allot 20 bytes for 1 letter terms.
  - And still can't handle *supercalifragilisticexpialidocious*.
- Written English averages ~4.5 characters.
  - Exercise: Why is/isn't this the number to use for estimating the dictionary size?
  - Short words dominate token counts.
- Average word in English: ~8 characters.

Explain this.

What are the corresponding numbers for Italian text?

Compressing the term list: Dictionary-as-a-String

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Hope to save up to 60% of dictionary space.



Total space for compressed list

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 3 bytes per term pointer
- Avg. 8 bytes per term in term string
- 500K terms ⇒ 9.5MB

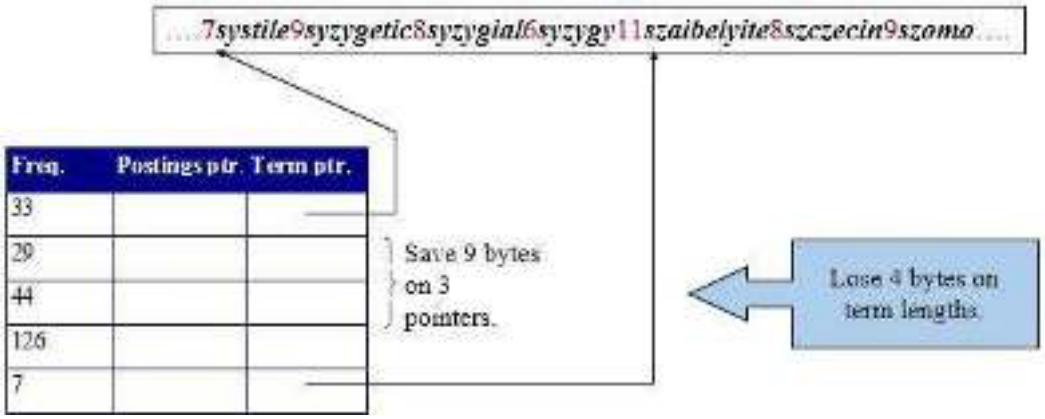
Total Space = 500K terms * (4 bytes Freq + 4 bytes Postings + 3 bytes term pointer + 8 bytes per term in term string)

Now avg. 11 bytes/term, not 20.

= 500K terms * (19 Bytes/Term)

Blocking

- Store pointers to every *k*th on term string.
    - Example below: *k*=4.
- Need to store term lengths (1 extra byte)



Net

- Where we used 3 bytes/pointer without blocking
    - 3 x 4 = 12 bytes for *k*=4 pointers,

now we use 3+4=7 bytes for 4 pointers.

Shaved another ~0.5MB; can save more with larger *k*.

Why not go with larger *k*?

Resources

- Chapter 5 of IIR
- Resources at http://ifnlp.org/ir
    - Original publication on word-aligned binary codes by Anh and Moffat (2005); also: Anh and Moffat (2006a)
    - Original publication on variable byte codes by Scholer, Williams, Yiannis and Zobel (2002)
    - More details on compression (including compression of positions and frequencies) in Zobel and Moffat (2006)

Lecture # 12

- Compression

ACKNOWLEDGEMENTS

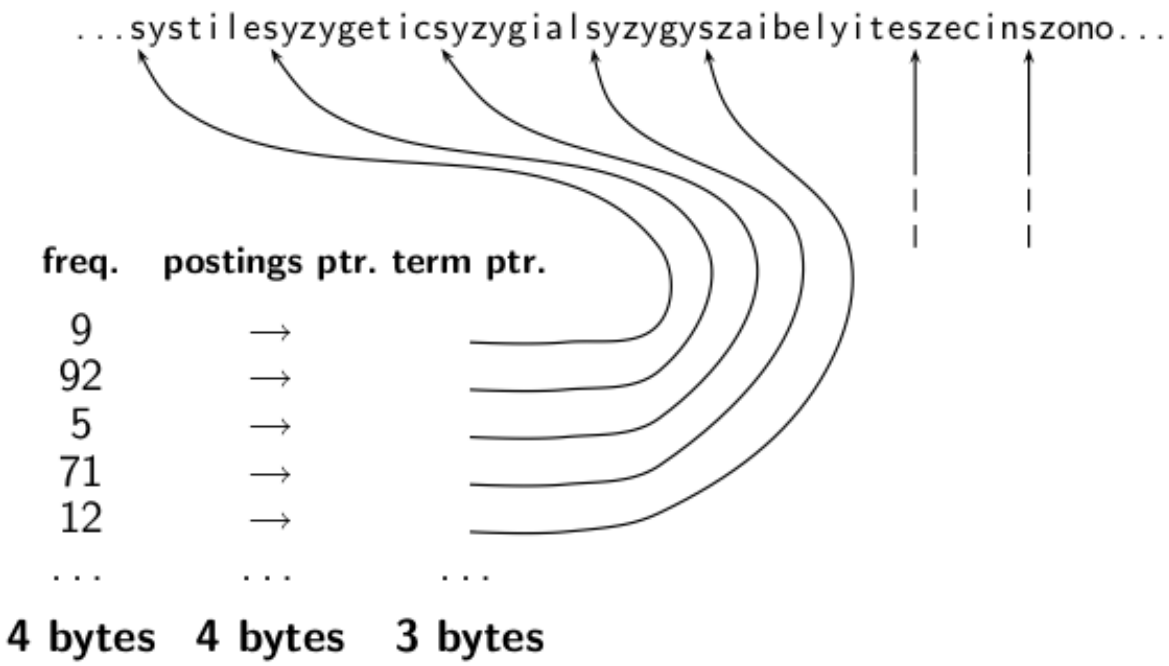
The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

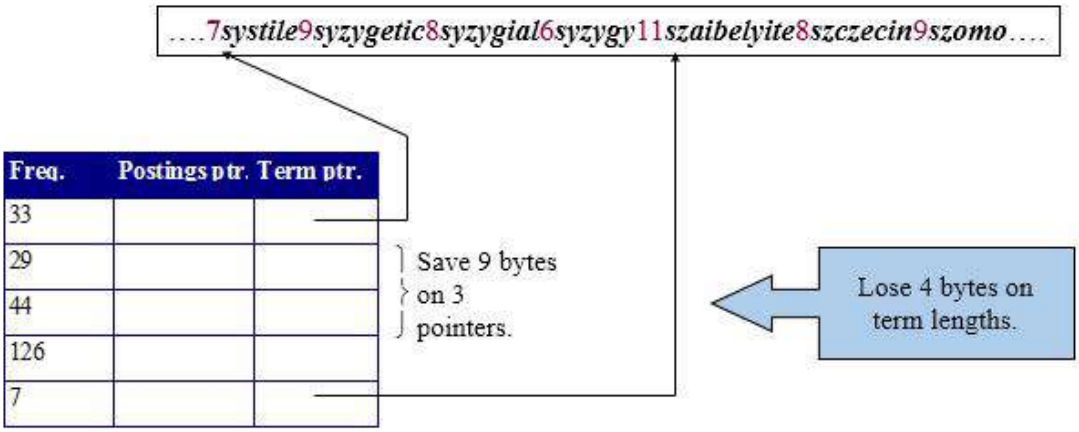
- Blocking
- Front coding
- Postings compression
- Variable Byte (VB) codes

Dictionary as a string



Blocking

- Store pointers to every *k*th on term string.
  - Example below: *k*=4.
- Need to store term lengths (1 extra byte)



Front coding

- Front-coding:
  - Sorted words commonly have long common prefix – store differences only
  - (for last *k-1* in a block of *k*)

8***automata***8***automate***9***automatic***10***automation***

→8 *automat** a1◊e2◊ic3◊ion

Encodes ***automat***

Extra length beyond ***automat.***

Begins to resemble general string compression.

Postings compression

- The postings file is much larger than the dictionary, factor of at least 10.
- Key desideratum: store each posting compactly
- A posting for our purposes is a docID.
- For Reuters (800,000 documents), we would use 32 bits per docID when using 4-byte integers.
- Alternatively, we can use $\log_2 800{,}000 \approx 19.6 < 20$ bits per docID.
- Our goal: use a lot less than 20 bits per docID.

Postings: two conflicting forces

- A term like ***arachnocentric*** occurs in maybe one doc out of a million – we would like to store this posting using $\log_2$ 1M ~ 20 bits.
- A term like ***the*** occurs in virtually every doc, so 20 bits/posting is too expensive.
    - Prefer 0/1 bitmap vector in this case

Postings file entry

- We store the list of docs containing a term in increasing order of docID.
    - ***computer***: 33,47,154,159,202 …
- Consequence: it suffices to store *gaps*.
    - 33,14,107,5,43 …
- Hope: most gaps can be encoded/stored with far fewer than 20 bits.

	encoding	postings list									
THE	docIDs	...		283042		283043		283044		283045	...
	gaps				1		1		1		...
COMPUTER	docIDs	...		283047		283154		283159		283202	...
	gaps				107		5		43		...
ARACHNOCENTRIC	docIDs	252000		500100							
	gaps	252000	248100								

Variable Byte (VB) codes

- For a gap value *G,* we want to use close to the fewest bytes needed to hold $\log_2 G$ bits
- Begin with one byte to store *G* and dedicate 1 bit in it to be a continuation bit *c*
- If *G* ≤127, binary-encode it in the 7 available bits and set *c* =1
- Else encode *G*'s lower-order 7 bits and then use additional bytes to encode the higher order bits using the same algorithm
- At the end set the continuation bit of the last byte to 1 (*c* =1) – and for the other bytes *c* = 0.

Example

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Postings stored as the byte concatenation
000001101011100010000101000011010000110010110001

Key property: VB-encoded postings are uniquely prefix-decodable.

For a small gap (5), VB uses a whole byte.

Resources

- Chapter 5 of IIR
- Resources at http://ifnlp.org/ir
  - Original publication on word-aligned binary codes by Anh and Moffat (2005); also: Anh and Moffat (2006a)
  - Original publication on variable byte codes by Scholer, Williams, Yiannis and Zobel (2002)
  - More details on compression (including compression of positions and frequencies) in Zobel and Moffat (2006)

Lecture # 13

- Compression

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Variable Byte (VB) codes
- Unary code
- Gamma codes
- Gamma code properties
- RCV1 compression

Variable Byte (VB) codes

- For a gap value *G,* we want to use close to the fewest bytes needed to hold $\log_2 G$ bits
- Begin with one byte to store *G* and dedicate 1 bit in it to be a <u>continuation</u> bit *c*
- If *G* ≤127, binary-encode it in the 7 available bits and set *c* =1
- Else encode *G*'s lower-order 7 bits and then use additional bytes to encode the higher order bits using the same algorithm
- At the end set the continuation bit of the last byte to 1 (*c* =1) – and for the other bytes *c* = 0.

Other variable unit codes

- Instead of bytes, we can also use a different "unit of alignment": 32 bits (words), 16 bits, 4 bits (nibbles).
- Variable byte alignment wastes space if you have many small gaps – nibbles do better in such cases.
- Variable byte codes:
    - Used by many commercial/research systems
    - Good low-tech blend of variable-length coding and sensitivity to computer memory alignment matches (vs. bit-level codes, which we look at next).
- There is also recent work on word-aligned codes that pack a variable number of gaps into one word

Unary code

- Represent *n* as *n* 1s with a final 0.
- Unary code for 3 is 1110.
- Unary code for 40 is

11111111111111111111111111111111111111110 .

- Unary code for 80 is:

11111111111111111111111111111111111111111111111111111111111111111111111111111111
110

- This doesn't look promising, but….

Gamma codes

- We can compress better with bit-level codes
  - The Gamma code is the best known of these.
- Represent a gap *G* as a pair *length* and *offset*
- *offset* is *G* in binary, with the leading bit cut off
  - For example 13 → 1101 → 101
- *length* is the length of offset
  - For 13 (offset 101), this is 3.
- We encode *length* with *unary code*: 1110.
- Gamma code of 13 is the concatenation of *length* and *offset*: 1110101

Gamma code examples

number	length	offset	γ-code
0			none
1	0		0
2	10	0	10,0
3	10	1	10,1
4	110	00	110,00
9	1110	001	1110,001
13	1110	101	1110,101
24	11110	1000	11110,1000
511	111111110	11111111	111111110,11111111
1025	11111111110	0000000001	11111111110,0000000001

Gamma code properties

- G is encoded using $2\lfloor \log G \rfloor + 1$ bits
    - Length of offset is $\lfloor \log G \rfloor$ bits
    - Length of length is $\lfloor \log G \rfloor + 1$ bits
- All gamma codes have an odd number of bits
- Almost within a factor of 2 of best possible, $\log_2 G$
- Gamma code is uniquely prefix-decodable, like VB
- Gamma code can be used for any distribution
- Gamma code is parameter-free

Reuters RCV1

symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types)	~400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
	non-positional postings	100,000,000

Gamma seldom used in practice

- Machines have word boundaries – 8, 16, 32, 64 bits
    - Operations that cross word boundaries are slower
- Compressing and manipulating at the granularity of bits can be slow
- Variable byte encoding is aligned and thus potentially more efficient
- Regardless of efficiency, variable byte is conceptually simpler at little additional space cost

Reuters RCV1

symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types)	~400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
	non-positional postings	100,000,000

RCV1 compression

Data structure	Size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
with blocking, k = 4	7.1
with blocking & front coding	5.9
collection (text, xml markup etc)	3,600.0
collection (text)	960.0
Term-doc incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
postings, γ–encoded	101.0

Index compression summary

- We can now create an index for highly efficient Boolean retrieval that is very space efficient
- Only 4% of the total size of the collection
- Only 10-15% of the total size of the text in the collection
- However, we've ignored positional information
- Hence, space savings are less for indexes used in practice

- But techniques substantially the same.

Resources

- Chapter 5 of IIR
- Resources at http://ifnlp.org/ir
    - Original publication on word-aligned binary codes by Anh and Moffat (2005); also: Anh and Moffat (2006a)
    - Original publication on variable byte codes by Scholer, Williams, Yiannis and Zobel (2002)
    - More details on compression (including compression of positions and frequencies) in Zobel and Moffat (2006)

Lecture # 14

- Index Constructions

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Scaling index construction
- Memory Hierarchy
- Hard Disk Tracks and Sectors
- Hard Disk Blocks
- Hardware basics

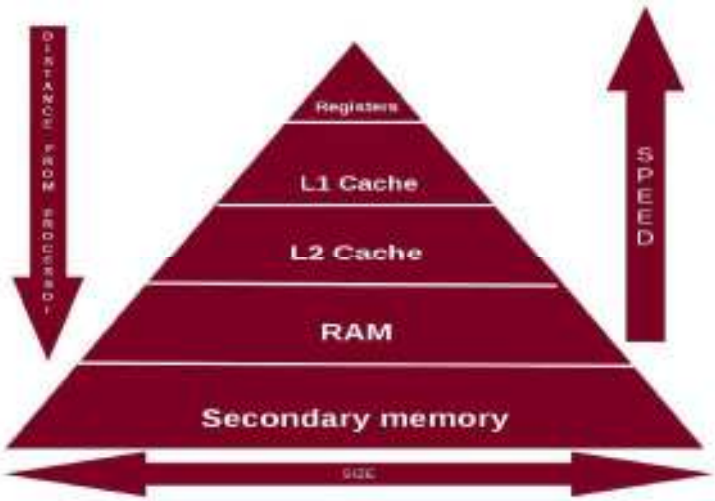
Scaling index construction

- In-memory index construction does not scale
  - Can't stuff entire collection into memory, sort, then write back
- How can we construct an index for very large collections?
- Taking into account the hardware constraints we just learned

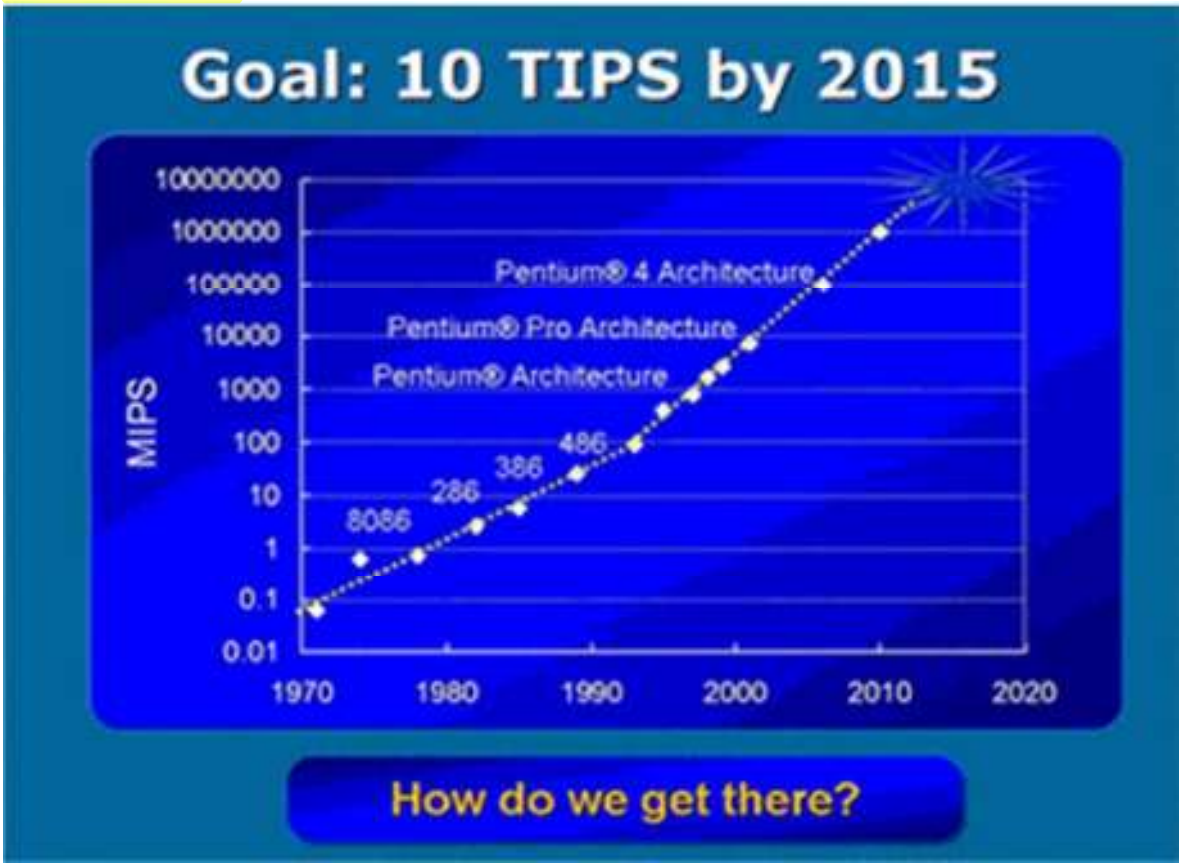
about . . .

- Memory, disk, speed, etc.

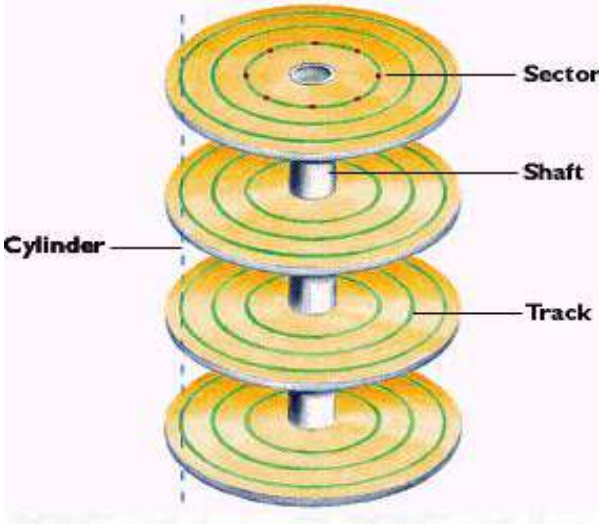
Memory Hierarchy



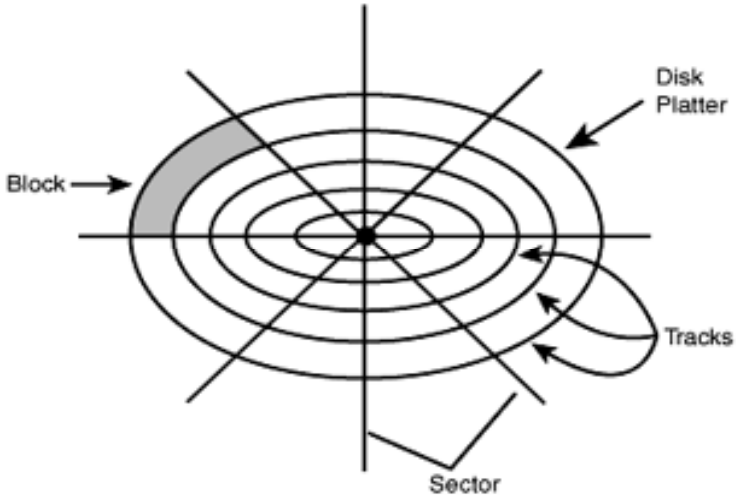
Moore's Law



Hard Disk Tracks and Sectors



Hard Disk Blocks



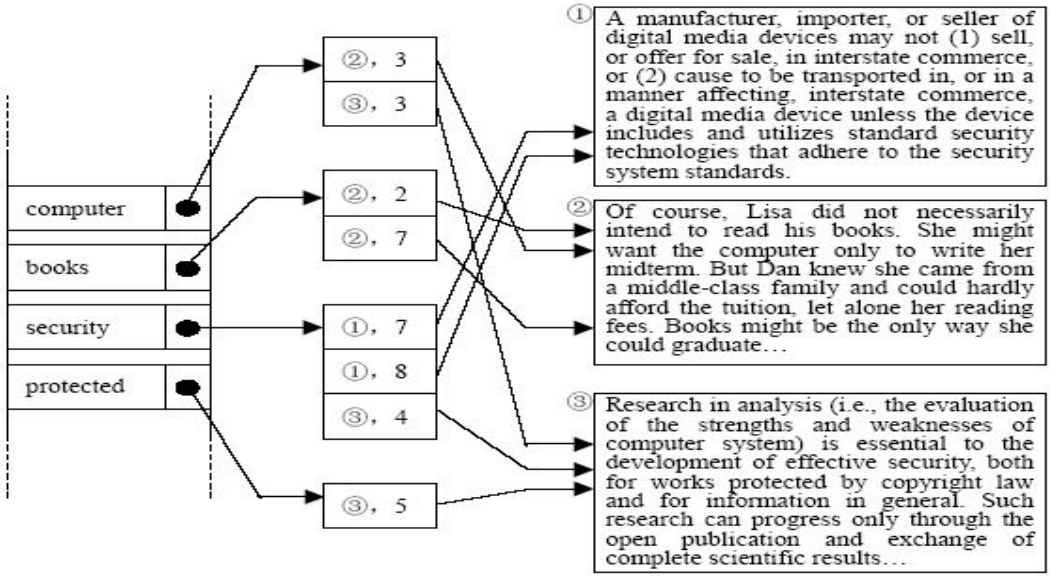
Disk Access Time

- Access time = (seek time) + (rotational delay) + (transfer time)
  - Seek time – moving the head to the right track
  - Rotational delay – wait until the right sector comes below the head
  - Transfer time – read/transfer the data

Hardware basics

- Access to data in memory is ***much*** faster than access to data on disk.
- Disk seeks: No data is transferred from disk while the disk head is being positioned.
- Therefore: Transferring one large chunk of data from disk to memory is faster than transferring many small chunks.
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks).
- Block sizes: 8KB to 256 KB.

Inverted Index



Inverted Index

Hardware basics (Cont….)

- Servers used in IR systems now typically have several GB of main memory, sometimes tens of GB.
- Available disk space is several (2–3) orders of magnitude larger.
- Fault tolerance is very expensive: It's much cheaper to use many regular machines rather than one fault tolerant machine.

Distributed computing

- Distributed computing is a field of computer science that studies distributed systems. A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages. Wikipedia

Hardware assumptions

symbol	statistic	value
s	average seek time	5 ms = 5×10^{-3} s
b	transfer time per byte	0.02 µs = 2×10^{-8} s
	processor's clock rate	10^{9} s^{-1}
p	low-level operation (e.g., compare & swap a word)	0.01 µs = 10^{-8} s
	size of main memory	several GB
	size of disk space	1 TB or more

Resources for today's lecture

- Chapter 4 of IIR
- MG Chapter 5
- Original publication on MapReduce: Dean and Ghemawat (2004)
- Original publication on SPIMI: Heinz and Zobel (2003)

Lecture # 15

- Merge Sort

ACKNOWLEDGEMENTS

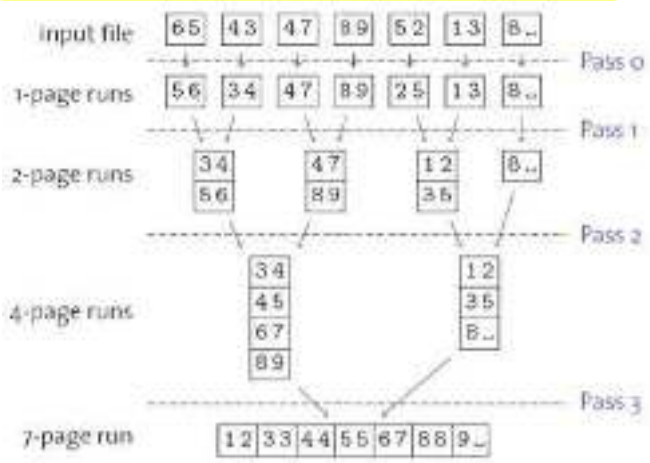
The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Two-Way Merge Sort
- Single-pass in-memory indexing
- SPIMI-Invert

Two-Way Merge Sort Example



Explain the improvement in 2 way merge by incorporating n-way merge

SPIMI:
Single-pass in-memory indexing

- Key idea 1: Generate separate dictionaries for each block – no need to maintain term-termID mapping across blocks.
- Key idea 2: Don't sort. Accumulate postings in postings lists as they occur.
- With these two ideas we can generate a complete inverted index for each block.
- These separate indexes can then be merged into one big index.

SPIMI-Invert

```
SPIMI-INVERT(token_stream)
output_file = NEWFILE()
dictionary = NEWHASH()
while (free memory available)
do token ← next(token_stream)
   if term(token) ∉ dictionary
      then postings_list = ADDTODICTIONARY(dictionary, term(token))
      else postings_list = GETPOSTINGSLIST(dictionary, term(token))
   if full(postings_list)
      then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
   ADDTOPOSTINGSLIST(postings_list, docID(token))
sorted_terms ← SORTTERMS(dictionary)
WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
return output_file
```

- Merging of blocks is analogous to BSBI.

Summary

- So far
    - Static Collections (corpus is fixed)
    - Linked List based indexing
    - Array based indexing
    - Data fits into the HD of a single machine
- Next
    - Data does not fit in a single machine
    - Requires more machines (clusters of machines).

Resources for today's lecture

- Chapter 4 of IIR
- MG Chapter 5
- Original publication on MapReduce: Dean and Ghemawat (2004)
- Original publication on SPIMI: Heinz and Zobel (2003)

Lecture # 16

- Phrase queries

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

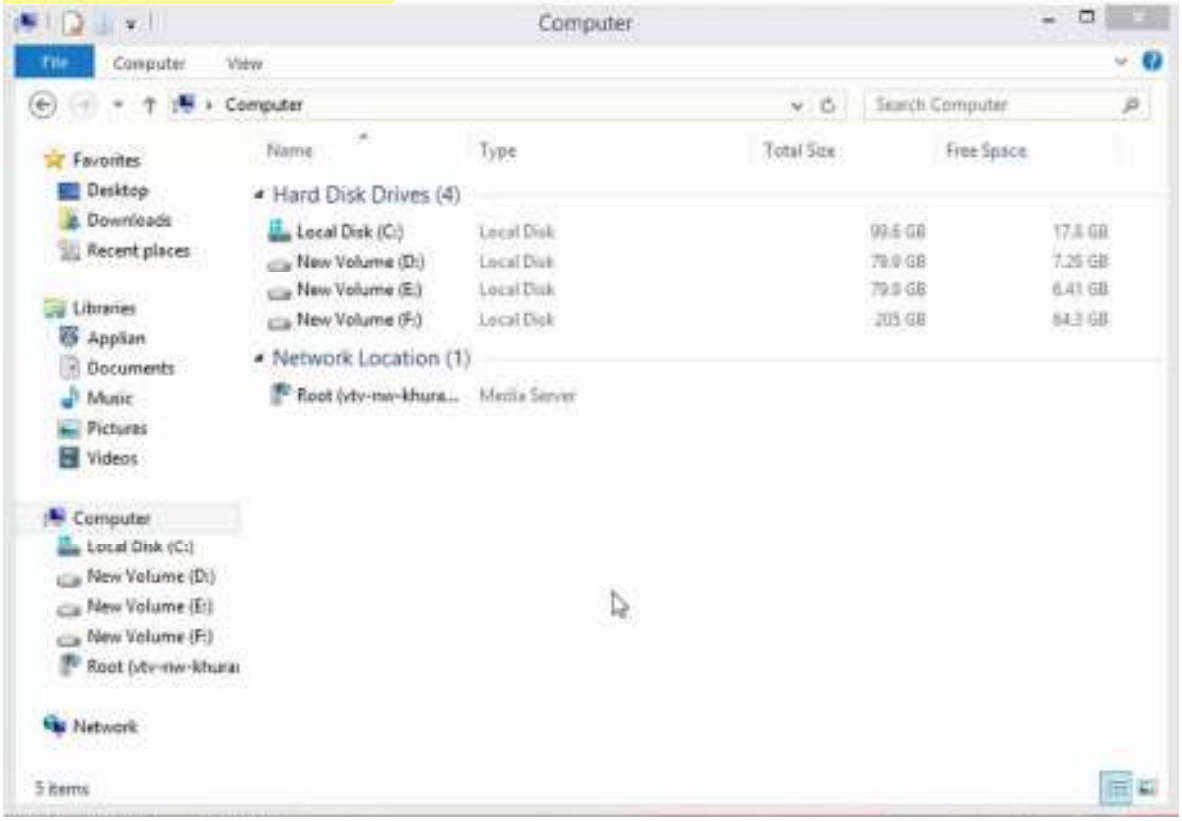
- Types of Queries
- Phrase queries
- Biword indexes
- Extended biwords
- Positional indexes

Click here to add title

Compiled by: Farhan Ahmed **VU ID:** MS160401398

Click here to add title

Click here to add title



Types of Queries

1. **Phrase Queries**

The crops in pakistan

"The crops in pakistan"

2. Proximity Queries

LIMIT! /3 STATUTE /3 FEDERAL /2 TORT

3. Wild Card Queries

Results*

Phrase queries

- Want to answer queries such as "***Stanford university***" – as a phrase
- Thus the sentence "I went to university at Stanford" is not a match.
- No longer suffices to store only

<*term* : *docs*> entries

A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text "Friends, Romans, Countrymen" would generate the biwords
    - ***friends romans***
    - ***romans countrymen***
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

Longer phrase queries

- Longer phrases are processed as we did with wild-cards:
- ***"stanford university palo alto"*** can be broken into the Boolean query on biwords:

"stanford university" *AND* ***"university palo"*** *AND* ***"palo alto"***

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Extended biwords

- Parse the indexed text and perform part-of-speech-tagging (POST).
- Bucket the terms into (say) Nouns (N) and articles/prepositions (X).
- Now deem any string of terms of the form NX*N to be an <u>extended biword</u>.
    - Each such extended biword is now made a term in the dictionary.
- Example:
    - ***catcher in the rye***
    - ***Capital of Pakistan***

Query processing

- Given a query, parse it into N's and X's
    - Segment query into enhanced biwords
    - Look up index
- Issues
    - Parsing longer queries into conjunctions
    - E.g., the query ***tangerine trees and marmalade skies*** is parsed into
    - ***tangerine trees*** *AND* ***trees and marmalade*** *AND* ***marmalade skies***

Other issues

- False positives, as noted before

- Index blowup due to bigger dictionary

Positional indexes

- Store, for each ***term***, entries of the form:

<number of docs containing ***term***;

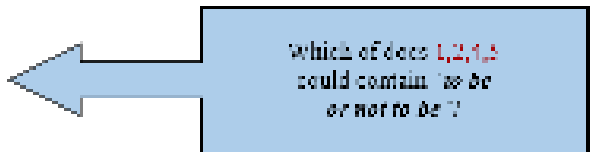
doc1: position1, position2 … ;

doc2: position1, position2 … ;

etc.>

Positional index example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>



- Can compress position values/offsets
- Nevertheless, this expands postings storage *substantially*

Resources for today's lecture

- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer: http//www.sims.berkeley.edu/~hearst/irbook/porter.html
- H.E. Williams, J. Zobel, and D. Bahle, "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.

http://www.seg.rmit.edu.au/research/research.php?author=4

Lecture # 17

- Processing a phrase query
- Proximity queries

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Processing a phrase query
- Proximity queries
- Combination schemes

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
    - ***to:***
        - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
    - ***be:***
        - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
- Same general method for proximity searches

Proximity queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT Here, /*k* means "within *k* words of".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of *k*?

Positional index size

- Can compress position values/offsets as we did with docs in the last lecture
- Nevertheless, this expands postings storage *substantially*

Positional index size

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
    - Average web page has <1000 terms
    - SEC filings, books, even some epic poems … easily 100,000 terms

Why?

- Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Rules of thumb

- Positional index size factor of 2-4 over non-positional index
- Positional index size 35-50% of volume of original text
- Caveat: all of this holds for "English-like" languages

Combination schemes

- A positional index expands postings storage *substantially (Why?)*
- Biword indexes and positional indexes approaches can be profitably combined
    - For particular phrases (***"Michael Jackson", "Britney Spears"***) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like ***"The Who"***

Wild Card Queries

- Example
- Stan* → Standard, Stanford
- S*T → Start
- *ion → Option
- Pa*an → Pakistan
- Pa*t*an etc…

Resources for today's lecture

- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer: http//www.sims.berkeley.edu/~hearst/irbook/porter.html
- H.E. Williams, J. Zobel, and D. Bahle, "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.

http://www.seg.rmit.edu.au/research/research.php?author=4

Lecture # 18

- Wild Card Queries
- B Tree

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- How to Handle Wild-Card Queries
- Wild-card queries: *
- B-Tree
- B+ Tree

How to Handle Wild-Card Queries

- B-Trees
- Permuterm Index
- K-Grams
- Soundex Algorithms

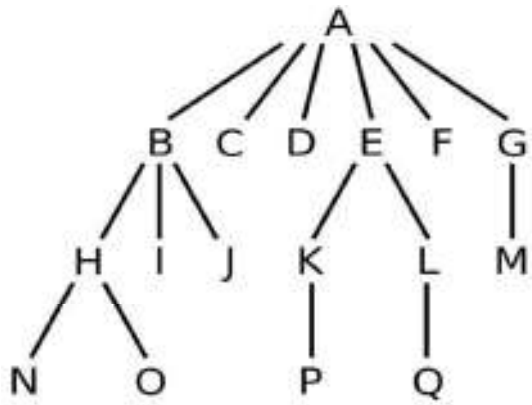
Wild-card queries: *

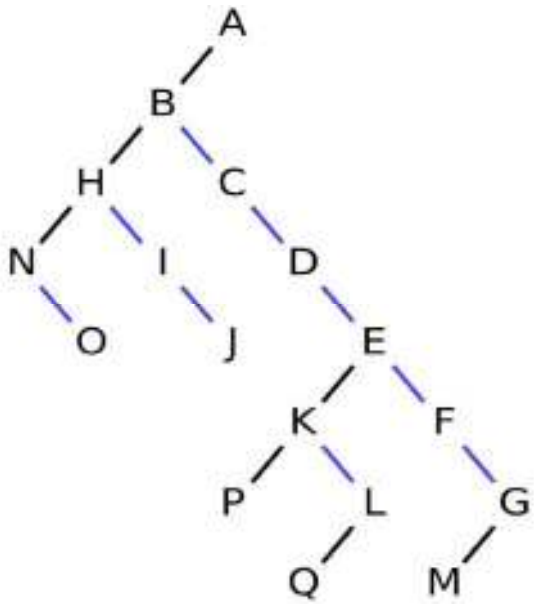
- ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ****mon:*** find words ending in "mon": harder
    - Maintain an additional B-tree for terms *backwards.*

Can retrieve all words in range: ***nom ≤ w < non***.

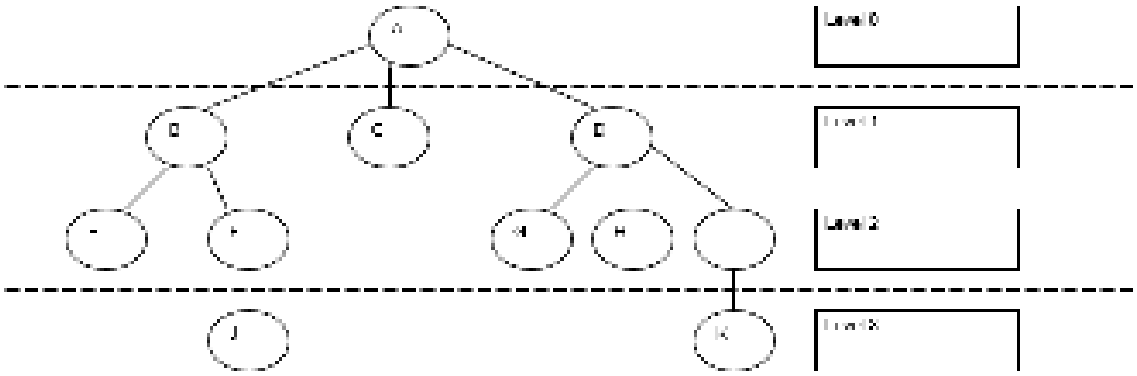
Exercise: from this, how can we enumerate all terms meeting the wild-card query ***pro*cent*** ?

B-Tree





B-Tree



n # of pairs.

# of external nodes = n + 1.

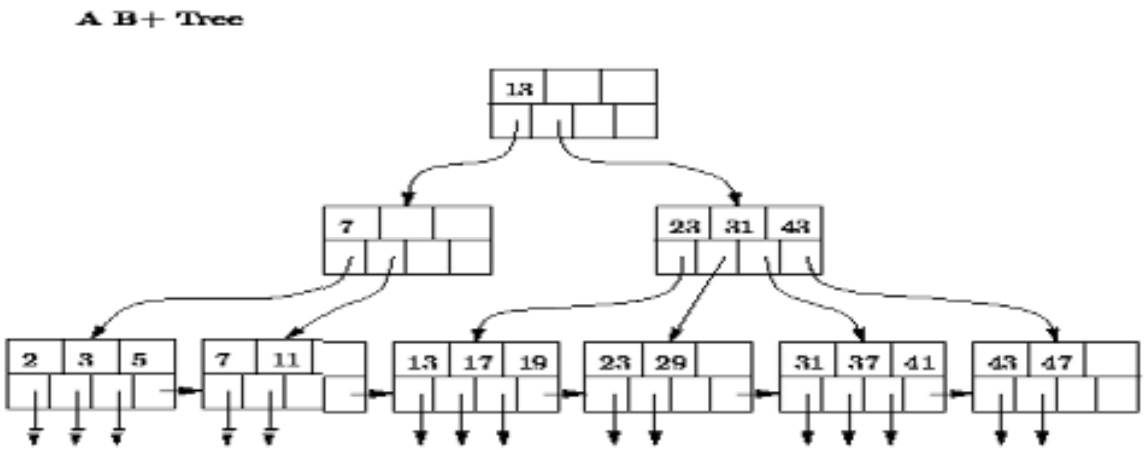
Wild-card queries: *

- ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ****mon:*** find words ending in "mon": harder
    - Maintain an additional B-tree for terms *backwards.*

Can retrieve all words in range: ***nom ≤ w < non.***

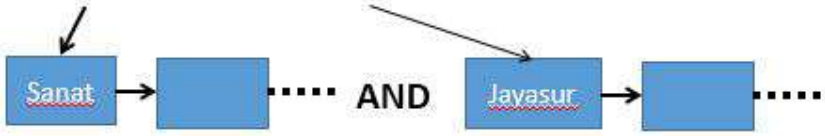
Exercise: from this, how can we enumerate all terms meeting the wild-card query ***pro*cent*** ?

B+ Tree



Wild-card queries example

- *Query*
  - *Sanat* AND Jayasur**



  - *Query*

    *mo*y*
    **day*

- Queries:
  - **X** lookup on **X$** — **X*** lookup on **$X***
  - ***X** lookup on **X$*** — ***X*** lookup on **X***
  - **X*Y** lookup on **Y$X*** — **X*Y*Z** ??? Exercise!

Resources

- IIR 3, MG 4.2

- Efficient spell retrieval:
  - K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
  - J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. http://citeseer.ist.psu.edu/zobel95finding.html
  - Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. http://citeseer.ist.psu.edu/179155.html
- **Nice, easy reading on spell correction:**
  - Peter Norvig: How to write a spelling corrector

http://norvig.com/spell-correct.html

Lecture # 19

- Permuterm index
- *k*-gram

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Permuterm index
- *k*-gram
- Soundex

Permuterm index

- For term ***hello***, index under:
  - ***hello$, ello$h, llohe, lohel, o$hell***

  where $ is a special symbol.
- Queries:
  - **X** lookup on **X$**
  - ***X** lookup on **X$***
  - **X*Y** lookup on **Y$X***

  X* lookup on **$X***

  X lookup on **X***

  X*Y*Z ??? Exercise!

a*e*l*o*u → e * u → u$a* → look up B tree

X, X*, *X are all simple and can be handled without permuterm.
X*Y and X*Y*Z can be handled with permuterm.

Query – hel*o
X=hel, Y=o
Lookup o$hel*

Permuterm index increases the size of dictionary 4 times
FOR ENGLISH
Depends on average length of a word

Bigram (*k*-gram) indexes

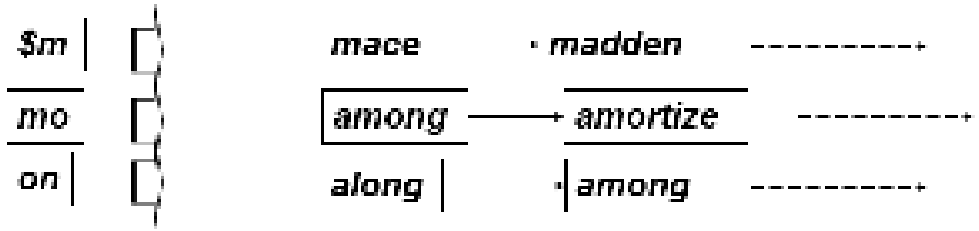
- Enumerate all *k*-grams (sequence of *k* chars) occurring in any term
- *e.g.*, from text "***April is the cruelest month***" we get the 2-grams (*bigrams*)

a,ap,pr,ri,il,l,i,is,s,t,th,he,e,$c,cr,ru,
ue,el,le,es,st,t$, m,mo,on,nt,h

  - $ is a special word boundary symbol
- Maintain a *second* inverted index *from bigrams to dictionary terms* that match each bigram.

Bigram index example

- The *k*-gram index finds *terms* based on a query consisting of *k*-grams (here *k*=2).



Processing wild-cards

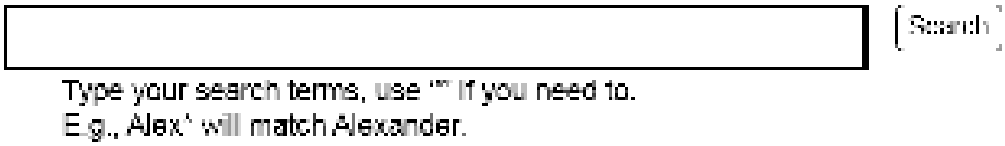
- Query ***mon**** can now be run as
  - ***$m*** *AND* ***mo*** *AND* ***on***
- Gets terms that match AND version of our wildcard query.
- But we'd enumerate ***moon***.
- Must post-filter these terms against query.
- Surviving enumerated terms are then looked up in the term-document inverted index.
- Fast, space efficient (compared to permuterm).

K-gram Index

- Larger k-grams would be lesser flexibility in query processing.
- Normally bi and tri k-gram proffered.

Processing wild-card queries

- As before, we must execute a Boolean query for each enumerated, filtered term.
- Wild-cards can result in expensive query execution (very large disjunctions…)
  - pyth* AND prog*
- If you encourage "laziness" people will respond!



- Which web search engines allow wildcard queries?

Soundex

- Class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for names
  - E.g., ***chebyshev*** → ***tchebycheff***
- Invented for the U.S. census … in 1918

Soundex – typical algorithm

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero): 'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Change letters to digits as follows:

- B, F, P, V → 1
- C, G, J, K, Q, S, X, Z → 2
- D,T → 3
- L → 4
- M, N → 5
- R → 6

e.g. Herman
H0rm0n
H0r505
H06505

Soundex continued

4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

E.g., ***Herman*** becomes H655.

Will ***hermann*** generate the same code?

Soundex

- Soundex is the classic algorithm, provided by most databases (Oracle, Microsoft, …)
- How useful is soundex?
- Generally used for Proper Nouns.
- Database systems also provide soundex option.
- Get names whose last name is same as soundex(SMITH)
- We may get Lindsay and William
- It lowers precision.
- But increases Recall.
- Can be used by Interpol to check names.
- Generally such algorithm are tailored according to European names.
- There are other variants which work well for IR.

Resources

- IIR 3, MG 4.2
- Efficient spell retrieval:
    - K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
    - J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. http://citeseer.ist.psu.edu/zobel95finding.html
    - Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. http://citeseer.ist.psu.edu/179155.html
- **Nice, easy reading on spell correction:**
    - Peter Norvig: How to write a spelling corrector

http://norvig.com/spell-correct.html

Lecture # 20

- Spelling Correction

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Spell correction
- Document correction
- Query mis-spellings
- Isolated word correction
- Edit distance
- Fibonacci series

Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Correcting user queries to retrieve "right" answers
- Two main flavors:
  - Isolated word
    - Check each word on its own for misspelling
    - Will not catch typos resulting in correctly spelled words
    - e.g., ***from*** → ***form***
  - Context-sensitive
    - Look at surrounding words,
    - e.g., ***I flew <u>form</u> Lahore to Dubai.***

Document correction

- Especially needed for OCR'ed documents
  - Correction algorithms are tuned for this: rn/m
    - **<u>modern</u>** may be considered as **<u>modem</u>**
  - Can use domain-specific knowledge
    - E.g., OCR can confuse O and D more often than it would confuse O and I (adjacent on the QWERTY keyboard, so more likely interchanged in typing).
- But also: web pages and even printed material have typos
- Goal: the dictionary contains fewer misspellings

- But often we don't change the documents and instead fix the query-document mapping

Query mis-spellings

- Our principal focus here
    - E.g., the query ***Fasbinder – Fassbinder (correct German Surname)***
- We can either
    - Retrieve documents indexed by the correct spelling, OR
    - Return several suggested alternative queries with the correct spelling
        - Google's *Did you mean … ?*

Query mis-spellings examples

Isolated word correction

- Fundamental premise – there is a lexicon from which the correct spellings come
- Two basic choices for this
    - A standard lexicon such as
        - Webster's English Dictionary
        - An "industry-specific" lexicon – hand-maintained
    - The lexicon of the indexed corpus
        - E.g., all words on the web
        - All names, acronyms etc.
        - (Including the mis-spellings)

Isolated word correction

- Given a lexicon and a character sequence Q, return the words in the lexicon closest to Q
- What's "closest"?
- We'll study several alternatives
    - Edit distance (Levenshtein distance)
    - Weighted edit distance
    - *n*-gram overlap

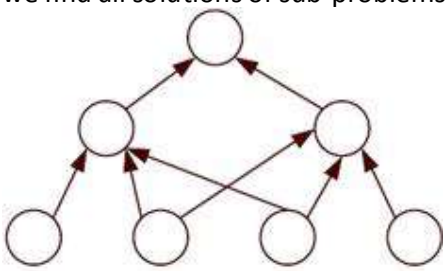
Edit distance

- Given two strings S_1 and S_2, the minimum number of operations to convert one to the other
- Operations are typically character-level
    - Insert, Delete, Replace, (Transposition)
- E.g., the edit distance from ***dof*** to ***dog*** is 1
    - From ***cat*** to ***act*** is 2 (Just 1 with transpose.)
    - from ***cat*** to ***dog*** is 3.
- Generally found by dynamic programming.
- See http://www.merriampark.com/ld.htm for a nice example plus an applet.

Problem Solving Techniques

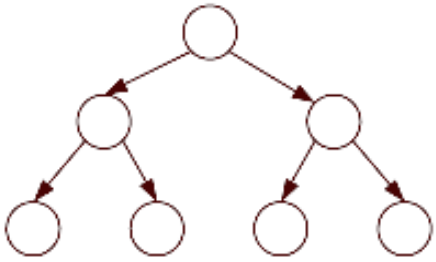
- **Dynamic programming:-**

Dynamic Programming solves the sub-problems bottom up. The problem can't be solved until we find all solutions of sub-problems. The solution comes up when the whole problem appears.



- **Divide and conquer:-**

Divide and Conquer works by dividing the problem into sub-problems, conquer each sub-problem recursively and combine these solutions.

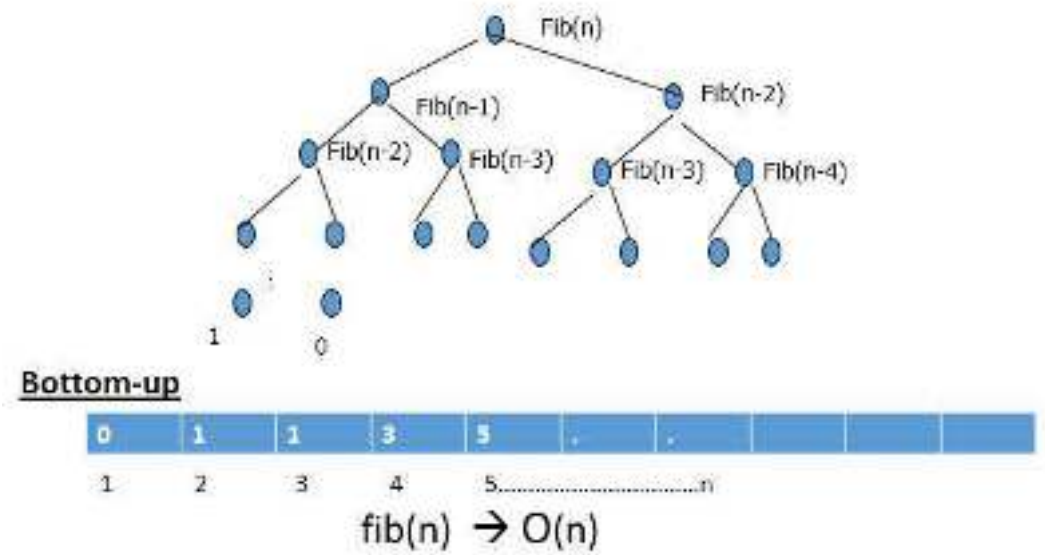


Fibonacci series

- **Definition:-** The first two numbers in the Fibonacci sequence are 1 and 1, or 0 and 1, depending on the chosen starting point of the sequence, and each subsequent number is the sum of the previous two(Wikipedia).

```
Fib(n)
{
    if (n == 1)
    return 0;
if (n == 2)
    return 1;
else
    return Fib(n-1) + Fib(n-2);
 }
```

Fibonacci Tree



Resources

- IIR 3, MG 4.2
- Efficient spell retrieval:
    - K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
    - J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. http://citeseer.ist.psu.edu/zobel95finding.html
    - Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. http://citeseer.ist.psu.edu/179155.html
- **Nice, easy reading on spell correction:**
    - Peter Norvig: How to write a spelling corrector

http://norvig.com/spell-correct.html

Lecture # 21

- Spelling Correction

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Edit distance
- Using edit distances
- Weighted edit distance
- *n*-gram overlap
- One option – Jaccard coefficient

Edit distance

- Given two strings S_1 and S_2, the minimum number of operations to convert one to the other
- Operations are typically character-level

Insert, Delete, Replace, (Transposition)

D(i,j) = score of best alignment from s1..si to t1..tj

$$= \min \begin{cases} D(i-1,j-1), \text{ if } s_i=t_j & \text{//copy} \\ D(i-1,j-1)+1, \text{ if } s_i \neq t_j & \text{//substitute} \\ D(i-1,j)+1 & \text{//insert} \\ D(i,j-1)+1 & \text{//Delete} \end{cases}$$

		m	o	n	k	e	y
	0	1	2	3	4	5	6
m	1	0	1	2	3	4	5
o	2	1	0	1	2	3	4
n	3	2	1	0	1	2	3
e	4	3	2	1	1	1	2
y	5	4	3	2	2	2	1

Using edit distances

- Given query, first enumerate all character sequences within a preset (weighted) edit distance (e.g., 2)
- Intersect this set with list of "correct" words
- Show terms you found to user as suggestions

- Alternatively,
    - We can look up all possible corrections in our inverted index and return all docs … slow
    - We can run with a single most likely correction
- The alternatives disempower the user, but save a round of interaction with the user

Weighted edit distance

- As before, but the weight of an operation depends on the character(s) involved
    - Meant to capture OCR or keyboard errors
      Example: ***m*** more likely to be mis-typed as ***n*** than as ***q***
    - Therefore, replacing ***m*** by ***n*** is a smaller edit distance than by ***q***
    - This may be formulated as a probability model
- Requires weight matrix as input
- Modify dynamic programming to handle weights

Weighted edit distance

- Case 1 :

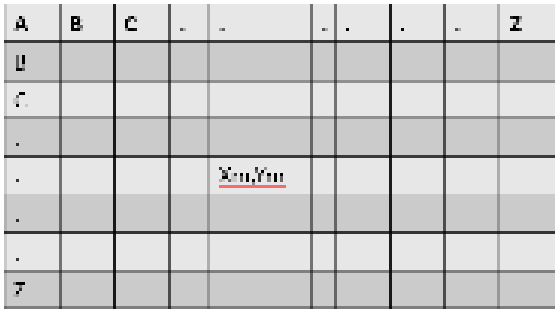
$E(m, n-1) + 1$

- Case 2 :

$E(m-1, n) + 1$

- Case 3:

$$E(m-1, n-1) + S = \begin{cases} 1 \text{ if } Xm \neq Ym \\ 0 \text{ if } Xm = Ym \end{cases}$$



Edit distance

- Given two strings S_1 and S_2, the minimum number of operations to convert one to the other
- Operations are typically character-level
    - Insert, Delete, Replace, (Transposition)
- E.g., the edit distance from ***dof*** to ***dog*** is 1
    - From ***cat*** to ***act*** is 2 (Just 1 with transpose.)
    - from ***cat*** to ***dog*** is 3.
- Generally found by dynamic programming.
- See http://www.merriampark.com/ld.htm for a nice example plus an applet.

n-gram overlap

- Enumerate all the *n*-grams in the query string as well as in the lexicon
- Use the *n*-gram index (recall wild-card search) to retrieve all lexicon terms matching any of the query *n*-grams
- Threshold by number of matching *n*-grams
    - Variants – weight by keyboard layout, etc.

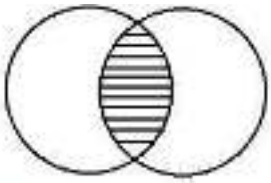
Example with trigrams

- Suppose the text is ***november***
    - Trigrams are *nov, ove, vem, emb, mbe, ber*.
- The query is ***december***
    - Trigrams are *dec, ece, cem, emb, mbe, ber*.
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?

One option – Jaccard coefficient

- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

$$|X \cap Y| / |X \cup Y|$$



- Equals 1 when *X* and *Y* have the same elements and zero when they are disjoint
- *X* and *Y* don't have to be of the same size
- Always assigns a number between 0 and 1
    - Now threshold to decide if you have a match
    - E.g., if J.C. > 0.8, declare a match

November: emb, mbe, ber
December:emb, mbe, ber

$|X \cap Y| = 3$
$|X \cup Y| = 3 + 3 + 3 = 9$
$JC = 3/9 = 1/3$

Resources

- IIR 3, MG 4.2
- Efficient spell retrieval:
    - K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
    - J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. http://citeseer.ist.psu.edu/zobel95finding.html

- Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. http://citeseer.ist.psu.edu/179155.html

- **Nice, easy reading on spell correction:**
  - Peter Norvig: How to write a spelling corrector

http://norvig.com/spell-correct.html

Lecture # 22

- Spelling Correction

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

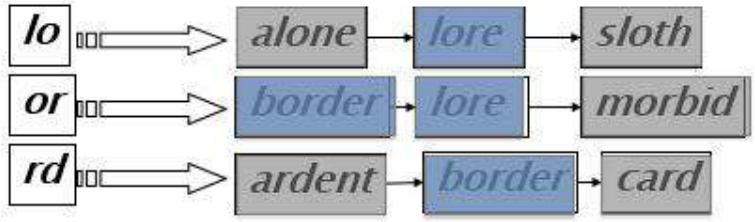
1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Matching trigrams
- Computing Jaccard coefficient
- Context-sensitive spell correction
- General issues in spell correction

Matching trigrams

- Consider the query ***lord*** – we wish to identify words matching 2 of its 3 bigrams (***lo, or, rd***)



Standard postings "merge" will enumerate …

Adapt this to using Jaccard (or another) measure.

2. ANSWER LIST

- Improve the walk through to use Jaccard Coefficient so as to identify the candidate terms.
- HEURISTIC
- While computing J.C. we may disregard the repeating n-grams in query term as well as in current term.

- The reason is that we are computing a candidate term in any case, which we shall process later using edit distance.

Computing Jaccard coefficient

$$|X \cap Y| / |X \cup Y|$$

X = Number of bigram in query tem

Y = Number of bigram in current term

Threshold: if J.C(Query), (Current term) > 0.8

append current term in answer list

$|X \cap Y|$ = Number of pointers at current term

$|X \cup Y| = |X| + |Y| - |X \cap Y|$

Joining N-grams with Edit Distance

- N-gram will give Answer list which contains candidate terms.
- These terms can then be checked for Edit Distance.
- This helps avoiding the checking of edit distance for all dictionary terms and each term, but restricts it to only candidate terms in the answer list

Context-sensitive spell correction

- Text: ***I flew <u>from</u> Lahore to Dubai.***
- Consider the phrase query ***"flew <u>form</u> Lahore"***
- We'd like to respond

  Did you mean "***flew from Lahore***"?

because no docs matched the query phrase.

Context-sensitive correction

- Need surrounding context to catch this.
    - NLP too heavyweight for this.
- First idea: retrieve dictionary terms close (in weighted edit distance) to each query term
- Now try all possible resulting phrases with one word "fixed" at a time
    - ***flew from heathrow***
    - ***fled form heathrow***
    - ***flea form heathrow***
    - ***etc.***
- Suggest the alternative that has lots of hits?
- Hits in corpus vs. Hits in Query Logs
    - It is more appropriate to look for hits in Qurey Logs

Context-sensitive correction

- Suppose that for ***"flew form Heathrow"*** we have 7 alternatives for flew, 19 for form and 3 for heathrow.
- 7*19*3 → Look for most frequent (in corpus/in query log) replacement of first word, combine it with the second to formulate a bigram, then choose the most frequent and then combine it with the third one. Reduces it to much less than 7*19*3
- Alternatively, **correct each word separately** that will result in 7+19+3
- Another alternative is to **only correct the misspelled words.**

General issues in spell correction

- We enumerate multiple alternatives for "Did you mean?"
- Need to figure out which to present to the user
    - The alternative hitting most docs
    - Query log analysis
- More generally, rank alternatives probabilistically

$$\text{argmax}_{corr}\, P(corr \mid query)$$

- From Bayes rule, this is equivalent to

$$\text{argmax}_{corr}\, P(query \mid corr) * P(corr)$$

Noisy channel Language model

Resources

- IIR 3, MG 4.2
- Efficient spell retrieval:

- K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
- J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. http://citeseer.ist.psu.edu/zobel95finding.html
- Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. http://citeseer.ist.psu.edu/179155.html

- **Nice, easy reading on spell correction:**
  - Peter Norvig: How to write a spelling corrector

http://norvig.com/spell-correct.html

Lecture # 23

- Performance Evaluation of Information Retrieval Systems

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Why System Evaluation?
- Difficulties in Evaluating IR Systems
- Measures for a search engine
- Measuring user happiness
- How do you tell if users are happy?

Why System Evaluation?

- There are many retrieval models/ algorithms/ systems, which one is the best?
- What is the best component for:
    - Ranking function (dot-product, cosine, …)
    - Term selection (stopword removal, stemming…)
    - Term weighting (TF, TF-IDF,…)
- How far down the ranked list will a user need to look to find some/all relevant documents?

Difficulties in Evaluating IR Systems

- Effectiveness is related to the ***relevancy*** of retrieved items.
- Relevancy is not typically binary but continuous.
- Even if relevancy is binary, it can be a difficult judgment to make.
- Relevancy, from a human standpoint, is:
    - Subjective: Depends upon a specific user's judgment.
    - Situational: Relates to user's current needs.
    - Cognitive: Depends on human perception and behavior.
    - Dynamic: Changes over time.

Measures for a search engine

- How fast does it index

    - Number of documents/hour
    - (Average document size)
- How fast does it search
    - Latency as a function of index size
- Expressiveness of query language
    - Ability to express complex information needs
    - Speed on complex queries
- Uncluttered UI
- Is it free?

Measuring user happiness

- Issue: who is the user we are trying to make happy?
    - Depends on the setting
- Web engine:
    - User finds what s/he wants and returns to the engine
        - Can measure rate of return users
    - User completes task – search as a means, not end
    - See Russell http://dmrussell.googlepages.com/JCDL-talk-June-2007-short.pdf
- eCommerce site: user finds what s/he wants and buys
    - Is it the end-user, or the eCommerce site, whose happiness we measure?
    - Measure time to purchase, or fraction of searchers who become buyers?

Measuring user happiness

- Enterprise (company/govt/academic): Care about "user productivity"
    - How much time do my users save when looking for information?
    - Many other criteria having to do with breadth of access, secure access, etc.

How do you tell if users are happy?

- Search returns products relevant to users
    - How do you assess this at scale?
- Search results get clicked a lot
    - Misleading titles/summaries can cause users to click
- Users buy after using the search engine
    - Or, users spend a lot of $ after using the search engine
- Repeat visitors/buyers
    - Do users leave soon after searching?
    - Do they come back within a week/month/… ?

Lecture # 24

- BENCHMARKS FOR THE EVALUATION OF IR SYSTEMS

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Happiness: elusive to measure
- Gold Standard
- search algorithm
- Relevance judgments
- Standard relevance benchmarks
- Evaluating an IR system

Happiness: elusive to measure

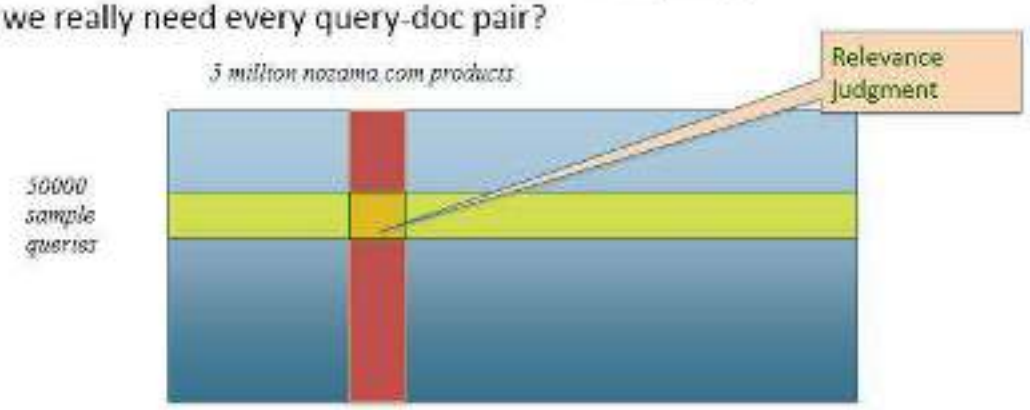
- Most common proxy: *relevance* of search results
- But how do you measure relevance?
- We will detail a methodology here, then examine its issues
- Relevance measurement requires 3 elements:
    1. A benchmark document collection
    2. A benchmark suite of queries
    3. A usually binary assessment of either Relevant or Nonrelevant for each query and each document
        - Some work on more-than-binary, but not the standard

Human Labeled Corpora (Gold Standard)

- Start with a corpus of documents.
- Collect a set of queries for this corpus.
- Have one or more human experts exhaustively label the relevant documents for each query.
- Typically assumes binary relevance judgments.
- Requires considerable human effort for large document/query corpora.

So you want to measure the quality of a new search algorithm

- Benchmark documents – nozama's products
- Benchmark query suite – more on this
- Judgments of document relevance for each query
  - Do we really need every query-doc pair?



Relevance judgments

- Binary (relevant vs. non-relevant) in the simplest case, more nuanced (0, 1, 2, 3 …) in others
- What are some issues already?
- 5 million times 50K takes us into the range of a quarter trillion judgments
  - If each judgment took a human 2.5 seconds, we'd still need 10^{11} seconds, or nearly $300 million if you pay people $10 per hour to assess
  - 10K new products per day

Crowd source relevance judgments?

- Present query-document pairs to low-cost labor on online crowd-sourcing platforms
  - Hope that this is cheaper than hiring qualified assessors
- Lots of literature on using crowd-sourcing for such tasks
- Main takeaway – you get some signal, but the variance in the resulting judgments is very high

What else?

- Still need test queries
  - Must be germane to docs available
  - Must be representative of actual user needs
  - Random query terms from the documents generally not a good idea
  - Sample from query logs if available
- Classically (non-Web)

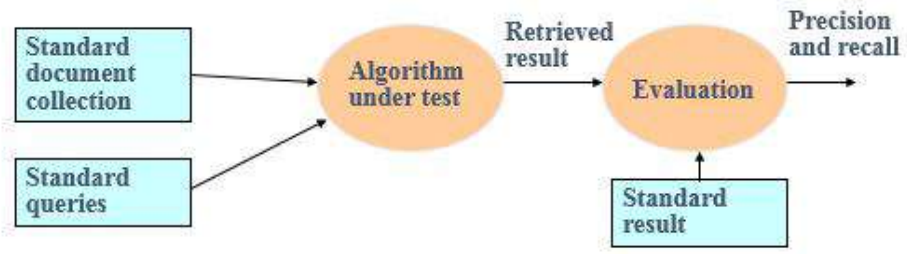
- Low query rates – not enough query logs
- Experts hand-craft "user needs"

Standard relevance benchmarks

- TREC (Text REtrieval Conference)- National Institute of Standards and Technology (NIST) has run a large IR test bed for many years
- Reuters and other benchmark doc collections used
- "Retrieval tasks" specified
    - sometimes as queries
- Human experts mark, for each query and for each doc, Relevant or Nonrelevant
    - or at least for subset of docs that some system returned for that query

Benchmarks

- A benchmark collection contains:
    - A set of standard documents and queries/topics.
    - A list of relevant documents for each query.
- Standard collections for traditional IR:
    - Smart collection: ftp://ftp.cs.cornell.edu/pub/smart
    - TREC: http://trec.nist.gov/



Some public test Collections

TABLE 4.3 Common Test Corpora

Collection	NDocs	NQrys	Size (MB)	Term/Doc	Q-D RelAss
ADI	82	35			
AIT	2109	14	2	400	>10,000
CACM	3204	64	2	24.5	
CISI	1460	112	2	46.5	
Cranfield	1400	225	2	53.1	
LISA	5872	35	3		
Medline	1033	30	1		
NPL	11,429	93	3		
OSHMED	34,8566	106	400	250	16,140
Reuters	21,578	672	28	131	
TREC	740,000	200	2000	89-3543	» 100,000

Typical TREC

Evaluating an IR system

- Note: **user need** is translated into a **query**
- Relevance is assessed relative to the **user need**, *not* the **query**
- E.g., Information need: *My swimming pool bottom is becoming black and needs to be cleaned.*
- Query: ***pool cleaner***
- Assess whether the doc addresses the underlying need, not whether it has these words

Lecture # 25

- BENCHMARKS FOR THE EVALUATION OF IR SYSTEMS

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

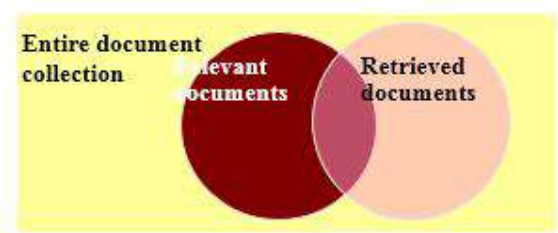
Outline

- Evaluation Measures
- Precision and Recall
- Unranked retrieval evaluation
- Trade-off between Recall and Precision
- Computing Recall/Precision Points

Evaluation Measures

- Precision
- Recall
- Accuracy
- Mean Average Precision
- F-Measure/E-Measure
- Non Binary Relevance
    - Discounted Cumulative Gain
    - Normalized Discounted Cumulative Gain

Precision and Recall



	retrieved	not retrieved
irrelevant	retrieved & irrelevant	Not retrieved & irrelevant
relevant	retrieved & relevant	not retrieved but relevant

$$recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents}$$

$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

Unranked retrieval evaluation: Precision and Recall

- **Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant)

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp/(tp + fp)
- Recall R = tp/(tp + fn)

Should we instead use the accuracy measure for evaluation?

- Given a query, an engine classifies each doc as "Relevant" or "Nonrelevant"
- The **accuracy** of an engine: the fraction of these classifications that are correct
  - ACCURACY = (tp + tn) / (tp + fp + fn + tn)

- **Accuracy** is a commonly used evaluation measure in machine learning classification work
- Why is this not a very useful evaluation measure in IR?

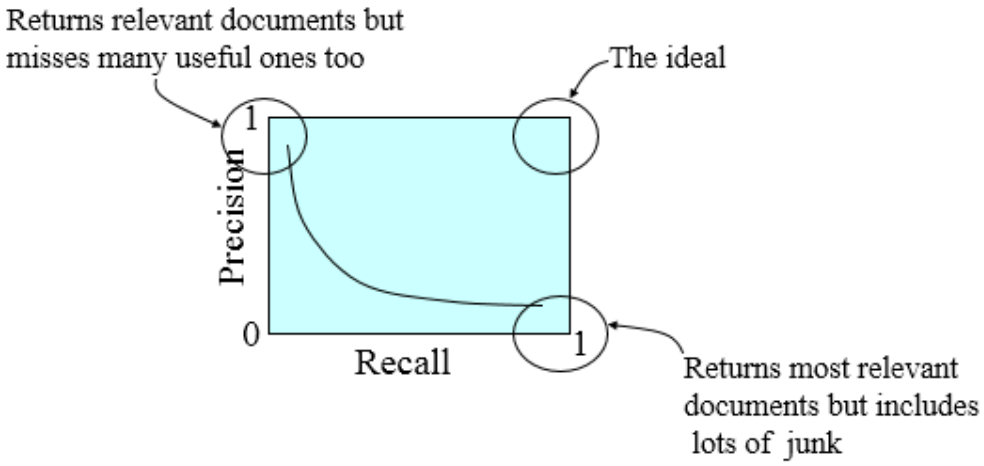
Precision and Recall

- Precision
  - The ability to retrieve top-ranked documents that are mostly relevant.
- Recall
  - The ability of the search to find ***all*** of the relevant items in the corpus.

Determining Recall is Difficult

- Total number of relevant items is sometimes not available:
  - Sample across the database and perform relevance judgment on these items.
  - Apply different retrieval algorithms to the same database for the same query. The aggregate of relevant items is taken as the total relevant set.

Trade-off between Recall and Precision



Computing Recall/Precision Points

- For a given query, produce the ranked list of retrievals.
- Adjusting a threshold on this ranked list produces different sets of retrieved documents, and therefore different recall/precision measures.
- Mark each document in the ranked list that is relevant according to the gold standard.
- Compute a recall/precision pair for each position in the ranked list that contains a relevant document.

Computing Recall/Precision Points: Example 1

n	doc #	relevant
1	588	x
2	589	x
3	576	
4	590	x
5	986	
6	592	x
7	984	
8	988	
9	578	
10	985	
11	103	
12	591	
13	772	x
14	990	

Let total # of relevant docs = 6
Check each new recall point:

R=1/6=0.167; P=1/1=1

R=2/6=0.333; P=2/2=1

R=3/6=0.5; P=3/4=0.75

R=4/6=0.667; P=4/6=0.667

R=5/6=0.833; p=5/13=0.38

Missing one relevant document. Never reach 100% recall

Lecture # 26

- Precision and Recall

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Computing Recall/Precision
- Interpolating a Recall/Precision
- Average Recall/Precision Curve
- R- Precision
- Precision@K
- F-Measure
- E Measure

Computing Recall/Precision Points: Example 1

n	doc #	relevant
1	588	x
2	589	x
3	576	
4	590	x
5	986	
6	592	x
7	984	
8	988	
9	578	
10	985	
11	103	
12	591	
13	772	x
14	990	

Let total # of relevant docs = 6
Check each new recall point:

R=1/6=0.167; P=1/1=1

R=2/6=0.333; P=2/2=1

R=3/6=0.5; P=3/4=0.75

R=4/6=0.667; P=4/6=0.667

R=5/6=0.833; p=5/13=0.38

Missing one relevant document. Never reach 100% recall

Computing Recall/Precision Points: Example 2

n	doc #	relevant
1	588	x
2	576	
3	589	x
4	342	
5	590	x
6	717	
7	984	
8	772	x
9	321	x
10	498	
11	113	
12	628	
13	772	
14	592	x

Let total # of relevant docs = 6
Check each new recall point:

R=1/6=0.167; P=1/1=1

R=2/6=0.333; P=2/3=0.667

R=3/6=0.5; P=3/5=0.6

R=4/6=0.667; P=4/8=0.5

R=5/6=0.833; P=5/9=0.556

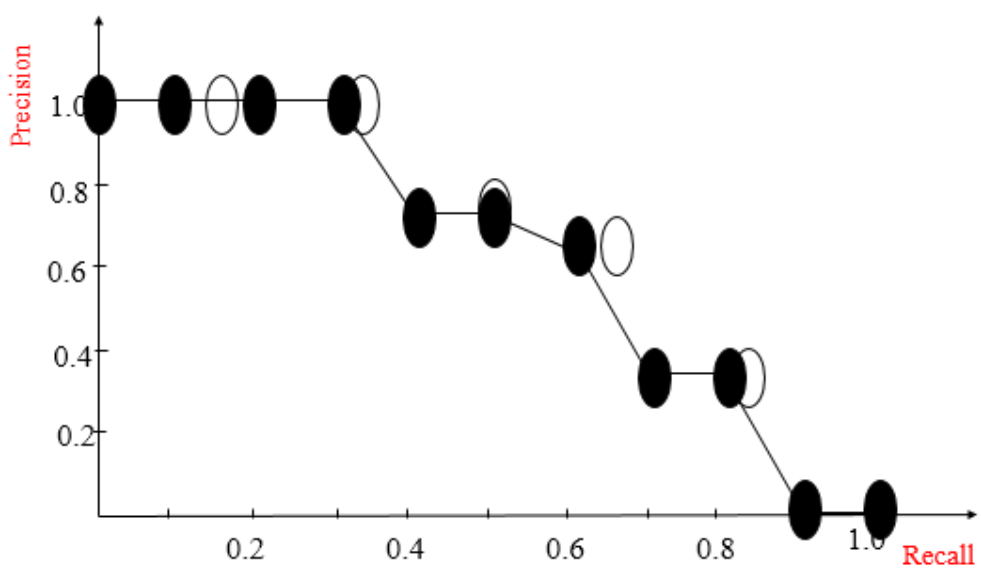
R=6/6=1.0; p=6/14=0.429

Interpolating a Recall/Precision Curve

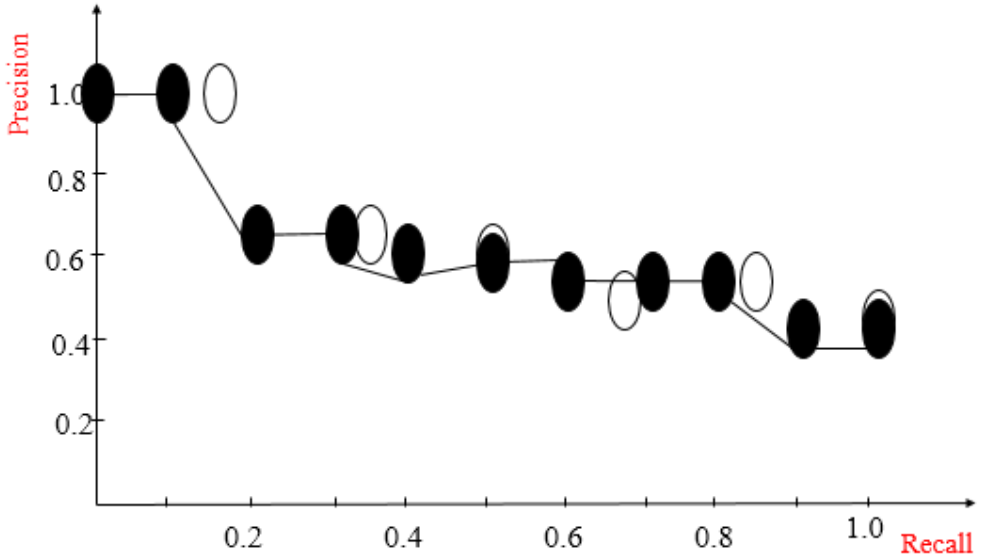
- Interpolate a precision value for each *standard recall level*:
  - r_j = {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  - r_0 = 0.0, r_1 = 0.1, …, r_{10}=1.0
- The interpolated precision at the *j*-th standard recall level is the maximum known precision at any recall level between the *j*-th and (*j* + 1)-th level:

$$P(r_j) = \max_{r_j \le r < r_{j+1}} P(r)$$

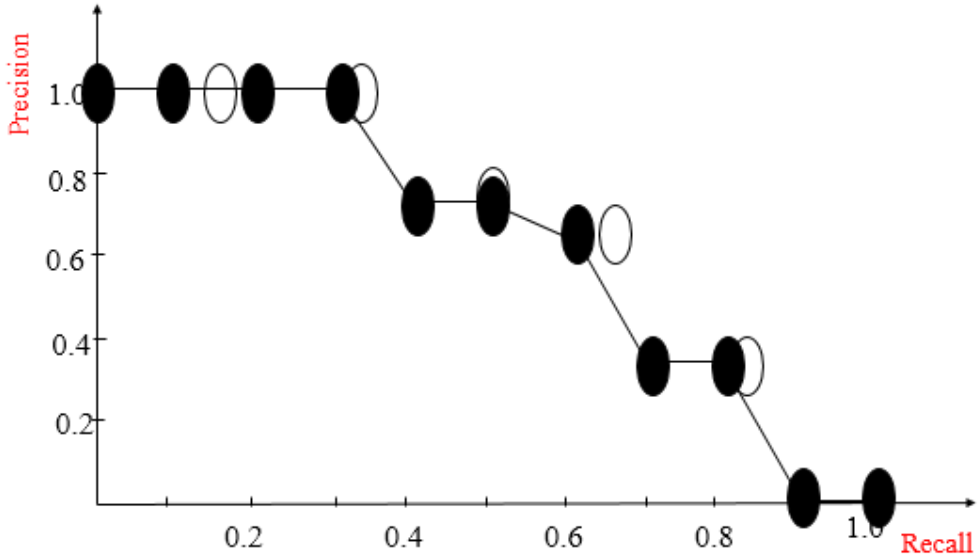
Interpolating a Recall/Precision Curve: Example 1



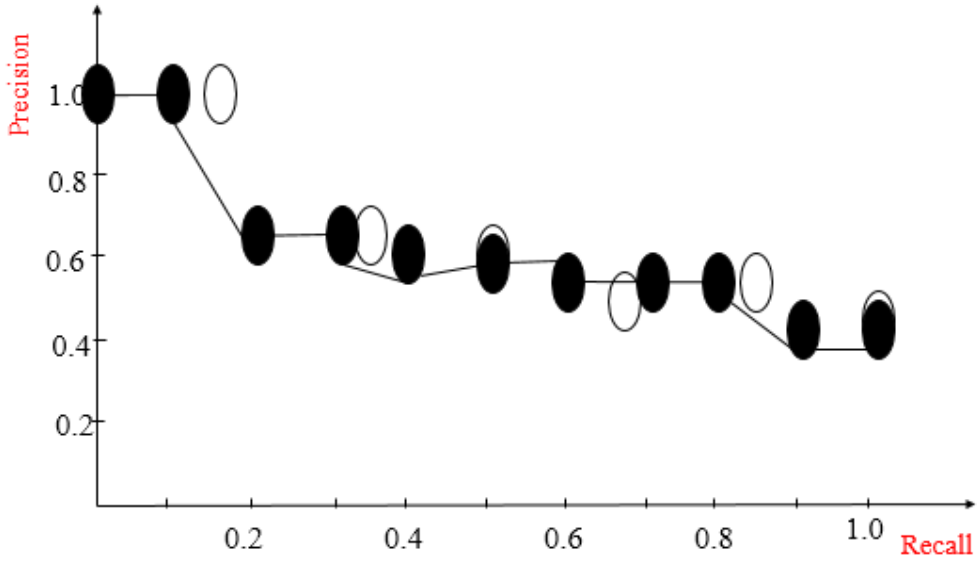
Interpolating a Recall/Precision Curve: Example 2



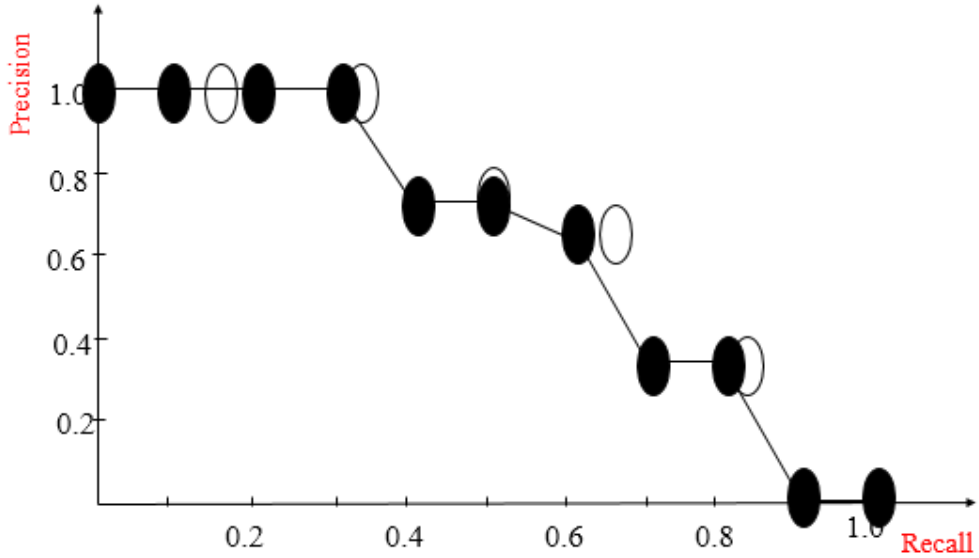
Interpolating a Recall/Precision Curve: Example 1



Interpolating a Recall/Precision Curve: Example 2



Interpolating a Recall/Precision Curve: Example 1

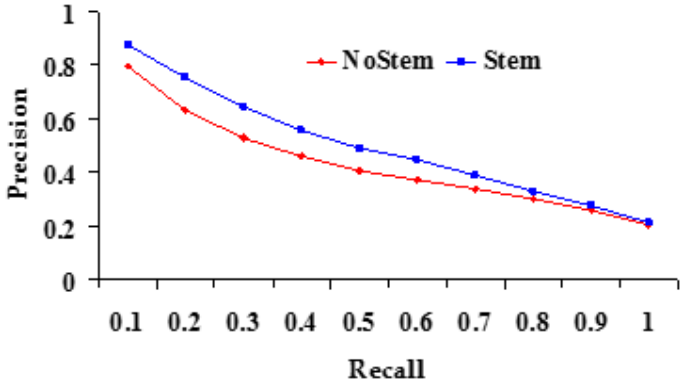


Average Recall/Precision Curve

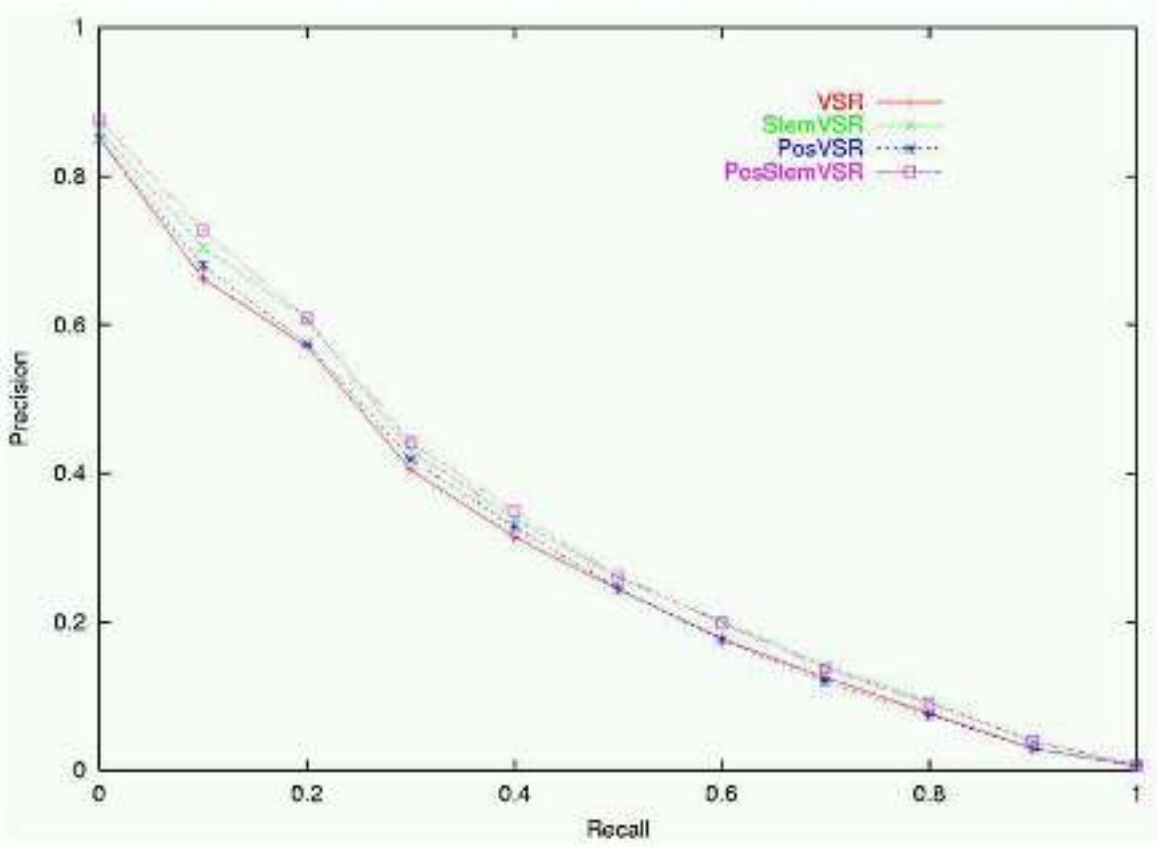
- Typically average performance over a large ***set*** of queries.
- Compute average precision at each standard recall level across all queries.
- Plot average precision/recall curves to evaluate overall system performance on a document/query corpus.

Compare Two or More Systems

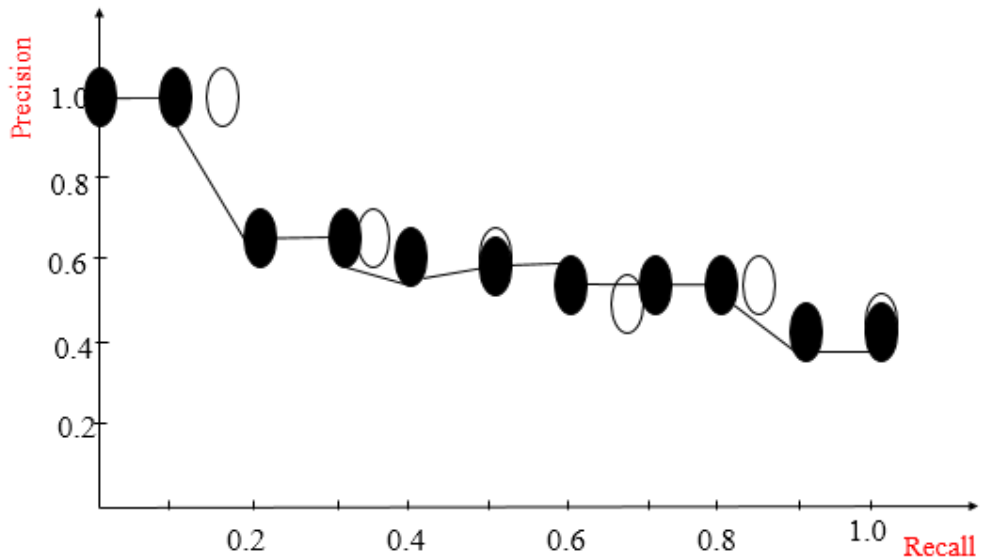
- The curve closest to the upper right-hand corner of the graph indicates the best performance



Sample RP Curve for CF Corpus



Interpolating a Recall/Precision Curve: Example 2



R- Precision

- Precision at the R-th position in the ranking of results for a query that has R relevant documents.

n	doc #	relevant
1	588	x
2	589	x
3	576	
4	590	x
5	986	
6	592	x
7	984	
8	988	
9	578	
10	985	
11	103	
12	591	
13	772	x
14	990	

R = # of relevant docs = 6

R-Precision = 4/6 = 0.67

Precision@K

- Set a rank threshold K
- Compute % relevant in top K
- Ignores documents ranked lower than K
- Ex:
  - Prec@3 of 2/3
  - Prec@4 of 2/4
  - Prec@5 of 3/5

- In similar fashion we have Recall@K

F-Measure

- One measure of performance that takes into account both recall and precision.
- Harmonic mean of recall and precision:

$$F = \frac{2PR}{P+R} = \frac{2}{\frac{1}{R}+\frac{1}{P}}$$

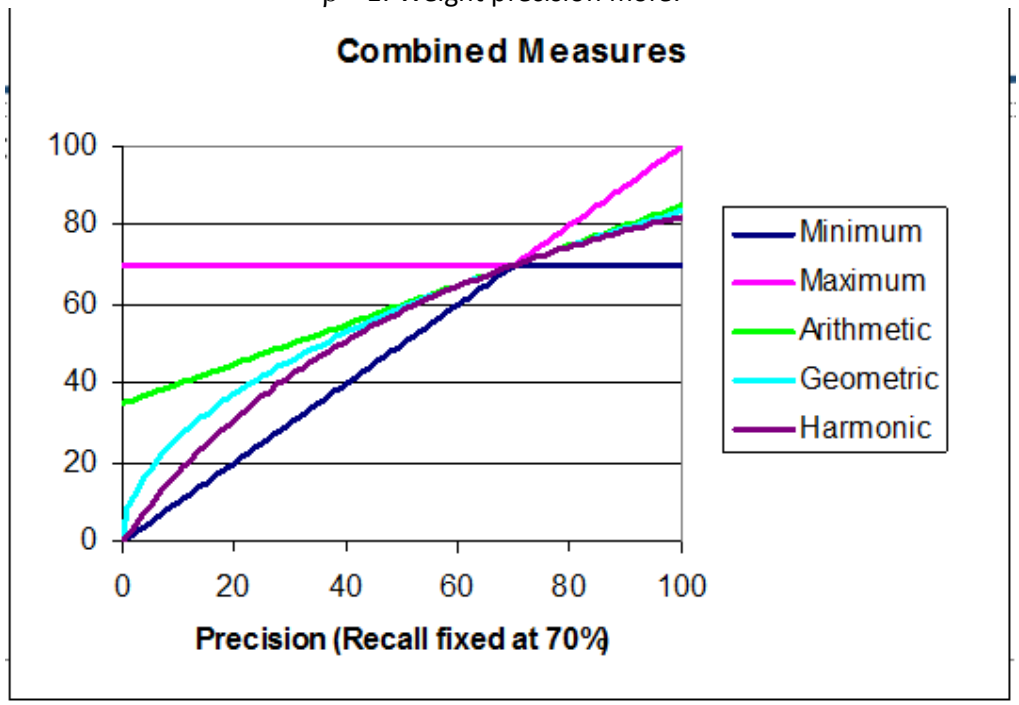
- Compared to arithmetic mean, both need to be high for harmonic mean to be high.

E Measure (parameterized F Measure)

- A variant of F measure that allows weighting emphasis on precision over recall:

$$E = \frac{(1+\beta^2)PR}{\beta^2 P + R} = \frac{(1+\beta^2)}{\frac{\beta^2}{R}+\frac{1}{P}}$$

- Value of β controls trade-off:
  - β = 1: Equally weight precision and recall (E=F).
  - β > 1: Weight recall more.
  - β < 1: Weight precision more.



Lecture # 27

- Mean Average Precision
- Non Binary Relevance
- DCG
- NDCG

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Mean Average Precision
- Mean Reciprocal Rank
- Cumulative Gain
- Discounted Cumulative Gain
- Normalized Discounted Cumulative Gain

Mean Average Precision (MAP)

- **Average Precision**: Average of the precision values at the points at which each relevant document is retrieved.
  - Ex1: (1 + 1 + 0.75 + 0.667 + 0.38 + 0)/6 = 0.633
  - Ex2: (1 + 0.667 + 0.6 + 0.5 + 0.556 + 0.429)/6 = 0.625
- **Mean Average Precision**: Average of the average precision value for a set of queries.

Mean average precision

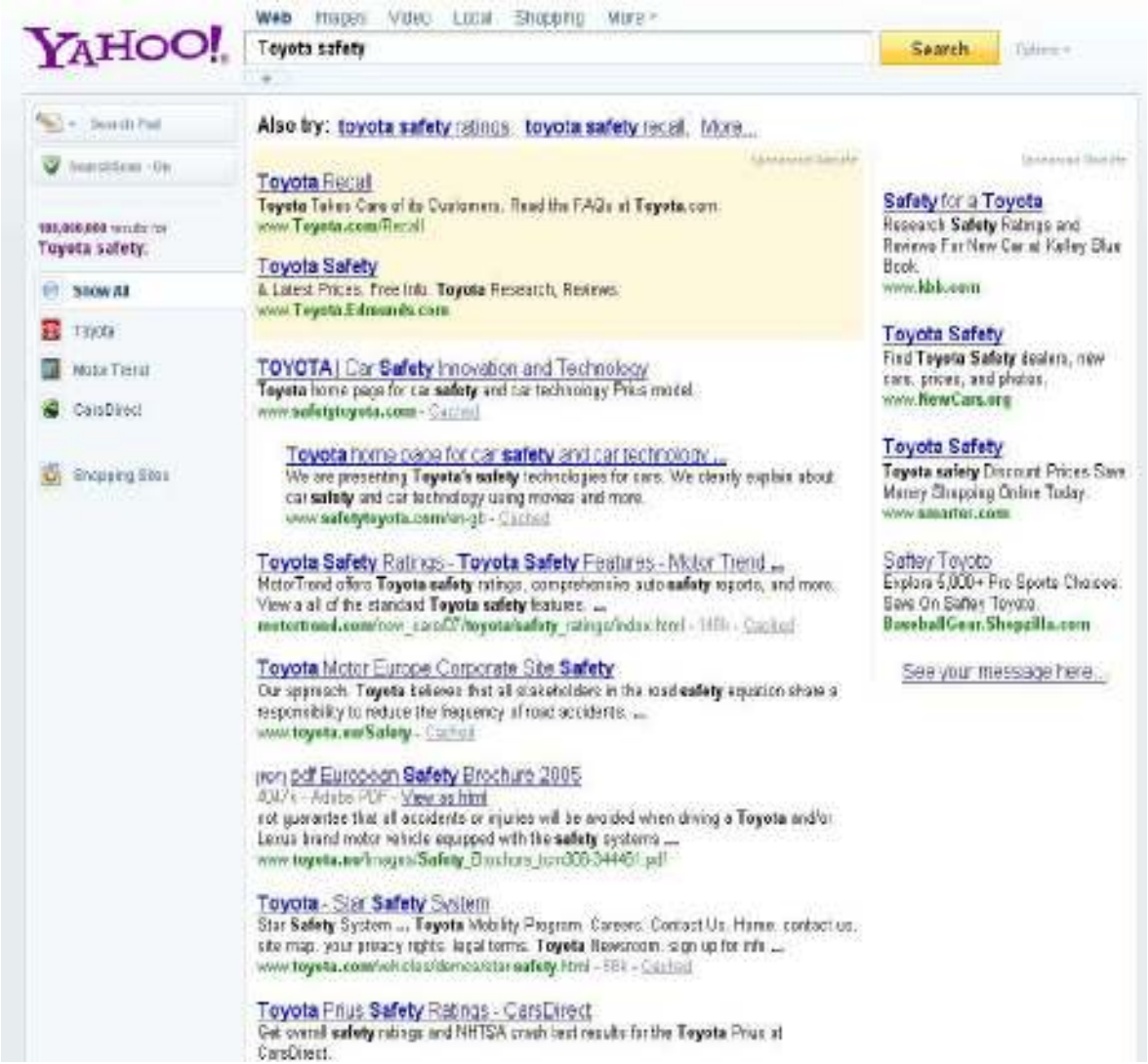
- If a relevant document never gets retrieved, we assume the precision corresponding to that relevant doc to be zero
- MAP is macro-averaging: each query counts equally
- Now perhaps most commonly used measure in research papers
- Good for web search?
- MAP assumes user is interested in finding many relevant documents for each query
- MAP requires many relevance judgments in text collection

Mean Reciprocal Rank

- Consider rank position, K, of first relevant doc
    - Could be – only clicked doc
    - Reciprocal Rank score =
- MRR is the mean RR across multiple queries

Non-Binary Relevance

- Documents are rarely entirely relevant or non-relevant to a query
- Many sources of *graded relevance judgments*
    - Relevance judgments on a 5-point scale
    - Multiple judges
    - Click distribution and deviation from expected levels (but click-through != relevance judgments)


YAHOO!
Web Images Video Local Shopping More
Toyota safety
Search
Also try: toyota safety ratings, toyota safety recall, More...
Toyota Recall
Toyota Safety
TOYOTA | Car Safety Innovation and Technology
Toyota home page for car safety and car technology ...
Toyota Safety Ratings - Toyota Safety Features - Motor Trend ...
Toyota Motor Europe Corporate Site Safety
Toyota - Star Safety System
Toyota Prius Safety Ratings - CarsDirect
Safety for a Toyota
Toyota Safety
Toyota Safety
Safety Toyota
See your message here...

Cumulative Gain

- With graded relevance judgments, we can compute the *gain* at each rank.
- **Cumulative Gain** at rank n:

$$CG_n = \sum_{i=1}^{n} rel_i$$

(Where rel_i is the graded relevance of the document at position *i)*

n	doc #	relevance (gain)	CG_n
1	588	1.0	1.0
2	589	0.6	1.6
3	576	0.0	1.6
4	590	0.8	2.4
5	986	0.0	2.4
6	592	1.0	3.4
7	984	0.0	3.4
8	988	0.0	3.4
9	578	0.0	3.4
10	985	0.0	3.4
11	103	0.0	3.4
12	591	0.0	3.4
13	772	0.2	3.6
14	990	0.0	3.6

Discounted Cumulative Gain

- Uses *graded relevance* as a measure of usefulness, or *gain,* from examining a document
- Gain is accumulated starting at the top of the ranking and may be reduced, or *discounted*, at lower ranks
- Typical discount is 1/log *(rank)*
    - With base 2, the discount at rank 4 is 1/2, and at rank 8 it is 1/3

Discounting Based on Position

- Users care more about high-ranked documents, so we **discount** results by *1/log$_2$(rank)*

- **Discounted Cumulative Gain:**

$$DCG_n = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{\log_2 i}$$

n	doc #	rel (gain)	CG_n	log_n	DCG_n
1	588	1.0	1.0	-	1.00
2	589	0.6	1.6	1.00	1.60
3	576	0.0	1.6	1.58	1.60
4	590	0.8	2.4	2.00	2.00
5	986	0.0	2.4	2.32	2.00
6	592	1.0	3.4	2.58	2.39
7	984	0.0	3.4	2.81	2.39
8	988	0.0	3.4	3.00	2.39
9	578	0.0	3.4	3.17	2.39
10	985	0.0	3.4	3.32	2.39
11	103	0.0	3.4	3.46	2.39
12	591	0.0	3.4	3.58	2.39
13	772	0.2	3.6	3.70	2.44
14	990	0.0	3.6	3.81	2.44

Normalized Discounted Cumulative Gain (NDCG)

- To compare DCGs, normalize values so that a *ideal ranking* would have a **Normalized DCG** of 1.0
- Ideal ranking:

Normalized Discounted Cumulative Gain (NDCG)

- To compare DCGs, normalize values so that a *ideal ranking* would have a **Normalized DCG** of 1.0
- Ideal ranking:

n	doc #	rel (gain)	CG_n	log_n	DCG_n
1	588	1.0	1.0	0.00	1.00
2	589	0.6	1.6	1.00	1.60
3	576	0.0	1.6	1.58	1.60
4	590	0.8	2.4	2.00	2.00
5	986	0.0	2.4	2.32	2.00
6	592	1.0	3.4	2.58	2.39
7	984	0.0	3.4	2.81	2.39
8	988	0.0	3.4	3.00	2.39
9	578	0.0	3.4	3.17	2.39
10	985	0.0	3.4	3.32	2.39
11	103	0.0	3.4	3.46	2.39
12	591	0.0	3.4	3.58	2.39
13	772	0.2	3.6	3.70	2.44
14	990	0.0	3.6	3.81	2.44

n	doc #	rel (gain)	CG_n	log_n	$IDCG_n$
1	588	1.0	1.0	0.00	1.00
2	592	1.0	2.0	1.00	2.00
3	590	0.8	2.8	1.58	2.50
4	589	0.6	3.4	2.00	2.80
5	772	0.2	3.6	2.32	2.89
6	576	0.0	3.6	2.58	2.89
7	986	0.0	3.6	2.81	2.89
8	984	0.0	3.6	3.00	2.89
9	988	0.0	3.6	3.17	2.89
10	578	0.0	3.6	3.32	2.89
11	985	0.0	3.6	3.46	2.89
12	103	0.0	3.6	3.58	2.89
13	591	0.0	3.6	3.70	2.89
14	990	0.0	3.6	3.81	2.89

Normalized Discounted Cumulative Gain (NDCG)

- Normalize by DCG of the ideal ranking:

$$NDCG_n = \frac{DCG_n}{IDCG_n}$$

- NDCG ≤ 1 at all ranks
- NDCG is comparable across different queries

n	doc #	rel (gain)	DCG_n	$IDCG_n$	$NDCG_n$
1	588	1.0	1.00	1.00	**1.00**
2	589	0.6	1.60	2.00	**0.80**
3	576	0.0	1.60	2.50	**0.64**
4	590	0.8	2.00	2.80	**0.71**
5	986	0.0	2.00	2.89	**0.69**
6	592	1.0	2.39	2.89	**0.83**
7	984	0.0	2.39	2.89	**0.83**
8	988	0.0	2.39	2.89	**0.83**
9	578	0.0	2.39	2.89	**0.83**
10	985	0.0	2.39	2.89	**0.83**
11	103	0.0	2.39	2.89	**0.83**
12	591	0.0	2.39	2.89	**0.83**
13	772	0.2	2.44	2.89	**0.84**
14	990	0.0	2.44	2.89	**0.84**

Lecture # 28

- Using user Clicks

ACKNOWLEDGEMENTS

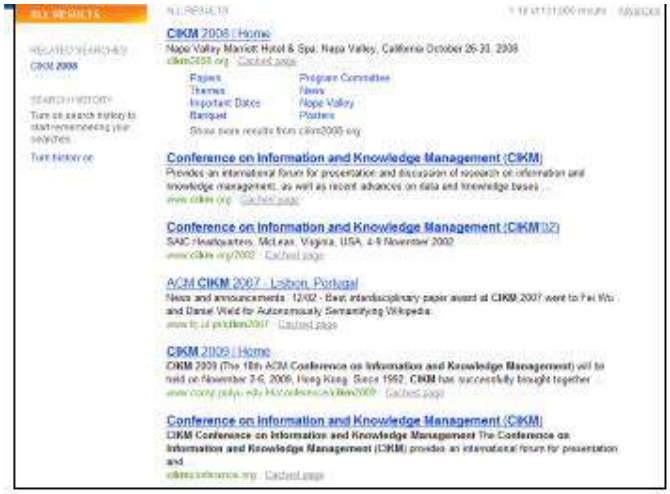
The presentation of this lecture has been taken from the underline sources

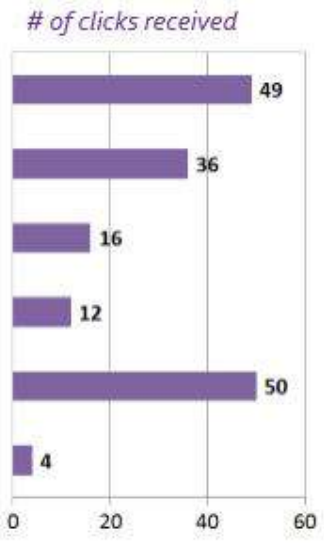
1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- What do clicks tell us?
- Relative vs absolute ratings
- Pairwise relative ratings
- Interleaved docs
- Kendall tau distance
- Critique of additive relevance
- Kappa measure
- A/B testing

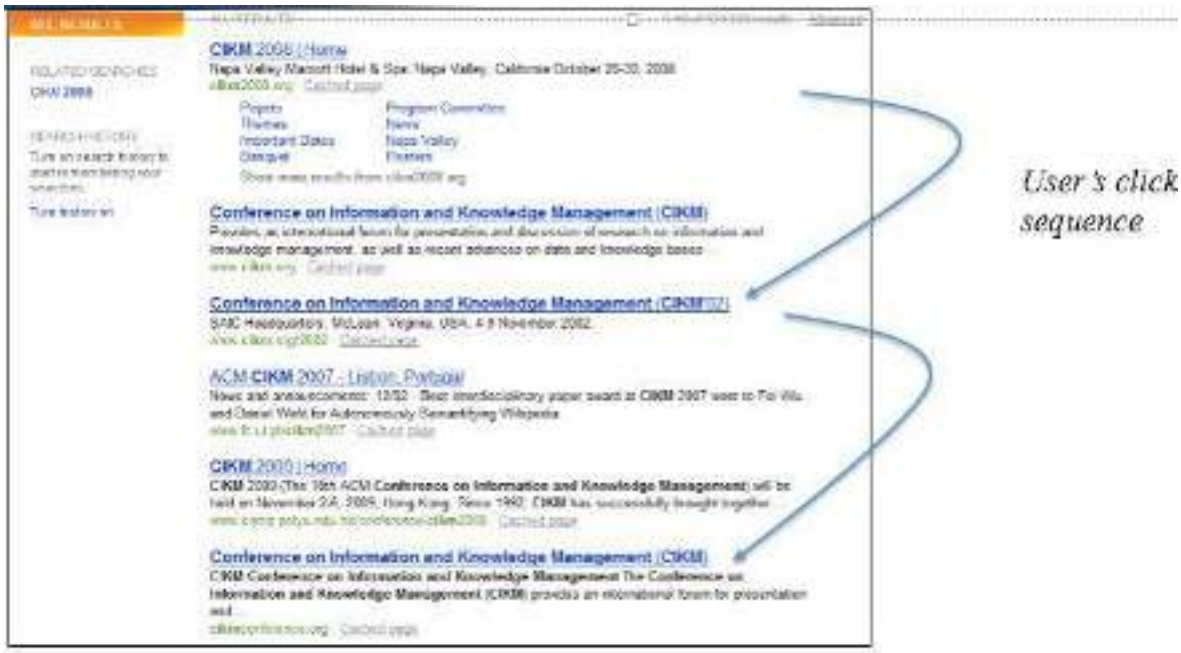
What do clicks tell us?





Strong position bias, so absolute click rates unreliable

Relative vs absolute ratings



Hard to conclude Result1 > Result3
Probably can conclude Result3 > Result2

Pairwise relative ratings

- Pairs of the form: DocA better than DocB for a query
    - Doesn't mean that DocA relevant to query
- Now, rather than assessing a rank-ordering wrt per-doc relevance assessments
- Assess in terms of conformance with historical pairwise preferences recorded from user clicks
- BUT!
- Don't learn and test on the same ranking algorithm
    - I.e., if you learn historical clicks from nozama and compare Sergey vs nozama on this history …

Interleaved docs (Joachims 2002)

- One approach is to obtain pairwise orderings from results that interleave two ranking engines A and B

Top From A	Top From B
Top From B	Top From A
2nd From A	2nd From B
2nd From B	2nd From A
3rd From A	3rd From B
3rd From B	3rd From A

Kendall tau distance

- Let X be the number of agreements between a ranking (say *A*) and *P*
- Let Y be the number of disagreements
- Then the Kendall tau distance between *A* and *P* is

(X-Y)/(X+Y)

- Say P = {(1,2), (1,3), (1,4), (2,3), (2,4), (3,4))} and A=(1,3,2,4)
- Then X=5, Y=1 …
- (What are the minimum and maximum possible values of the Kendall tau distance?)

Critique of additive relevance

- Relevance vs Marginal Relevance
    - A document can be redundant even if it is highly relevant
        - Duplicates
        - The same information from different sources
    - Marginal relevance is a better measure of utility for the user
        - But harder to create evaluation set
        - See Carbonell and Goldstein (1998)
    - Pushes us to assess a *slate* of results, rather than to sum relevance over individually assessed results
        - Raters shown two lists, and asked to pick the better one
        - Reminiscent of interleaved doc idea we just saw

Kappa measure for inter-judge (dis)agreement

- Kappa measure
    - Agreement measure among judges
    - Designed for categorical judgments
    - Corrects for chance agreement
- Kappa = [P(A) – P(E)] / [1 – P(E)]

- P(A) – proportion of time judges agree
- P(E) – what agreement would be by chance
- Kappa = 0 for chance agreement, 1 for total agreement.

Kappa Measure: Example

Number of docs	Judge 1	Judge 2
300	Relevant	Relevant
70	Nonrelevant	Nonrelevant
20	Relevant	Nonrelevant
10	Nonrelevant	Relevant

Kappa Example

- P(A) = 370/400 = 0.925
- P(nonrelevant) = (10+20+70+70)/800 = 0.2125
- P(relevant) = (10+20+300+300)/800 = 0.7878
- P(E) = 0.2125^2 + 0.7878^2 = 0.665
- Kappa = (0.925 – 0.665)/(1-0.665) = 0.776
- Kappa > 0.8 = good agreement
- 0.67 < Kappa < 0.8 -> "tentative conclusions" (Carletta '96)
- Depends on purpose of study
- For >2 judges: average pairwise kappas

A/B testing

- Purpose: Test a single innovation
- Prerequisite: You have a large search engine up and running.
- Have most users use old system
- Divert a small proportion of traffic (e.g., 1%) to the new system that includes the innovation
- Evaluate with an "automatic" measure like clickthrough on first result
- Now we can directly see if the innovation does improve user happiness.

- Judge effectiveness by measuring change in ***clickthrough:*** The percentage of users that click on the top result (or any result on the first page).
- Probably the evaluation methodology that large search engines trust most
- In principle less powerful than doing a multivariate regression analysis, but easier to understand

Lecture # 29

- Cosine ranking

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

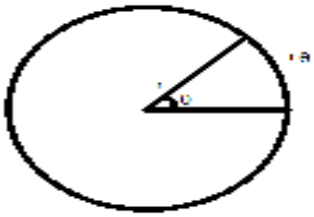
- Computing cosine-based ranking
- Efficient cosine ranking
- Computing the *K* largest cosines
- Use heap for selecting top *K*
- High-idf query terms

Computing cosine-based ranking

- Speeding up cosine ranking
- reducing the number of cosine computations
    - Union of term-wise candidates
    - Sampling and pre-grouping
- reducing the number of dimensions
    - Random projection
    - Latent semantic indexing

Efficient cosine ranking

- What we're doing in effect: solving the *K*-nearest neighbor problem for a query vector
- In general, we do not know how to do this efficiently for high-dimensional spaces
- But it is solvable for short queries, and standard indexes support this well



Computing the *K* largest cosines: selection vs. sorting

- Typically we want to retrieve the top *K* docs (in the cosine ranking for the query)
  - not to totally order all docs in the collection
- Can we pick off docs with *K* highest cosines?
- Let *J* = number of docs with nonzero cosines
  - We seek the *K* best of these *J*

Measuring the distance between 2 vectors

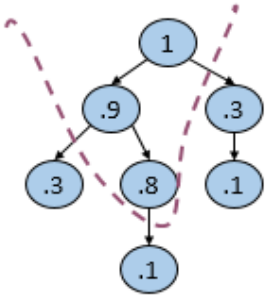
- **Measuring the distance between 2 vectors in V dimensional space is based on the angles between the vectors, where we have unit length for each vector.**
- **Here in 2D, we can see a circle, and the angle θ is from inside and**
- **r x θ is the difference as well.**

Computing the *K* largest cosines: selection vs. sorting

- Typically we want to retrieve the top *K* docs (in the cosine ranking for the query)
  - not to totally order all docs in the collection
- Can we pick off docs with *K* highest cosines?
- Let *J* = number of docs with nonzero cosines
  - We seek the *K* best of these *J*

Use heap for selecting top *K*

- Binary tree in which each node's value > the values of children
- Takes *2J* operations to construct, then each of *K* "winners" read off in 2log *J* steps.
- For *J*=1M, *K*=100, this is about 10% of the cost of sorting.



Use heap for selecting top *K (cont….)*

- Build Heap Takes O(n) time.
- Delete max element takes constant time, but fixing the rest of the heap takes log(n) time.
- Replacing 'n' with 'j' would result into O(J).

- Each Deletion would take log(J) time.
- For 'K' such operations we need K log(J) operations.

Use heap for selecting top *K (cont….)*

Typically we want to retrieve the top k docs

(in the cosine ranking for the query)

- Not totally order all docs in the corpus.
- Just pick off docs with k highest cosines.

Cosine similarity is only a proxy

- User has a task and a query formulation
- Cosine matches docs to query
- Thus cosine is anyway a proxy for user happiness
- If we get a list of *K* docs "close" to the top *K* by cosine measure, should be ok

Generic approach

- Find a set *A* of *contenders*, with $K < |A| << N$
    - *A* does not necessarily contain the top *K,* but has many docs from among the top *K*
    - Return the top *K* docs in *A*
- Think of *A* as pruning non-contenders
- The same approach is also used for other (non-cosine) scoring functions
- Will look at several schemes following this approach

Index elimination

- Basic algorithm cosine computation algorithm only considers docs containing at least one query term
- Take this further:
    - Only consider high-idf query terms
    - Only consider docs containing many query terms

High-idf query terms only

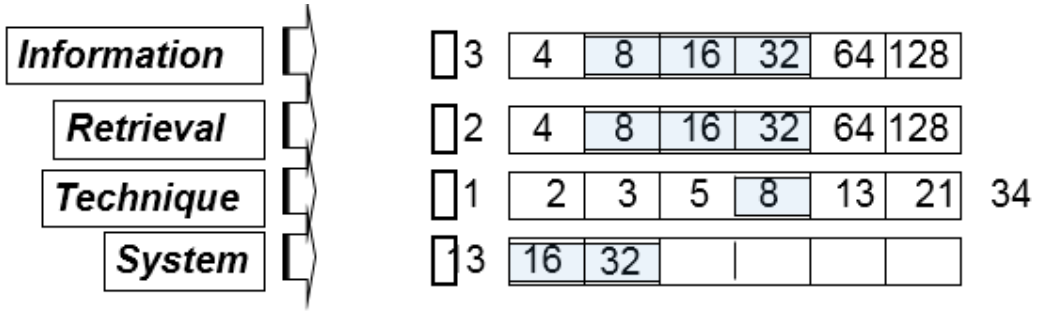
- For a query such as *catcher in the rye*
- Only accumulate scores from *catcher* and *rye*
- Intuition: ***in*** and ***the*** contribute little to the scores and so don't alter rank-ordering much
- Benefit:

- Postings of low-idf terms have many docs → these (many) docs get eliminated from set *A* of contenders

Docs containing many query terms

- Any doc with at least one query term is a candidate for the top *K* output list
- For multi-term queries, only compute scores for docs containing several of the query terms
    - Say, at least 3 out of 4
    - Imposes a "soft conjunction" on queries seen on web search engines (early Google)
- Easy to implement in postings traversal

3 of 4 query terms



Scores only computed for docs 8, 16 and 32.

High-idf query terms only

- For a query such as *catcher in the rye*
    - Only accumulate scores from *catcher* and *rye*
    - Intuition: ***in*** and ***the*** contribute little to the scores and so <u>don't alter rank-ordering much</u>
    - Benefit:
        - Postings of low-idf terms have many docs → these (many) docs get eliminated from set *A* of contenders

Lecture # 30

- Sampling and pre-grouping

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Term-wise candidates
- Preferred List storage
- Sampling and pre-grouping
- General variants

Term-wise candidates

Preprocess: Pre-compute, for each term, its *k* nearest docs.

- (Treat each term as a 1-term query.)
- lots of preprocessing.
- Result: "preferred list" for each term.

Search:

- For a t-term query, take the union of their *t* preferred lists - call this set *S.*
- Compute cosines from the query to only the docs in *S,* and choose top *k.*

Preferred List storage:

- **Store postings in descending order of similarity to the term**
- **Would disturb the postings merging, as it won't be stored in a sorted order of DocID**

- **Store separate postings list, which is preferred list**
  - **Store the postings in the ascending order of DocID**
  - **Would keep the merging of posting lists easy**
  - **Requires extra storage**
- **Number of Required Results**
- **In case the preferred lists are not able to produce required number of top results, go to the normal postings lists.**
- **Generate estimated number of total results against a query by using total number of postings against the terms**
- **Produce top results by using preferred lists**

- An example where a result among top-k is not reported

	K = 4			
Information	D5/0.85	D7/0.82	D3/0.81	D9/0.80
Retrieval	D7/0.9	D1/0.85	D6/0.82	D5/0.81
Techniques	D9/0.95	D7/0.90	D6/0.90	D12/0.88
Result	D7	D9	D6	D5

- An example where a result among top-k is not reported

	K = 4				
Information	D5/0.85	D7/0.82	D3/0.81	D9/0.80	D8/0.75
Retrieval	D7/0.9	D1/0.85	D6/0.82	D5/0.81	D8/0.8
Techniques	D9/0.95	D7/0.90	D6/0.90	D12/0.88	D8/0.88
Result	D7	D9	D6	D5	

Sampling and pre-grouping

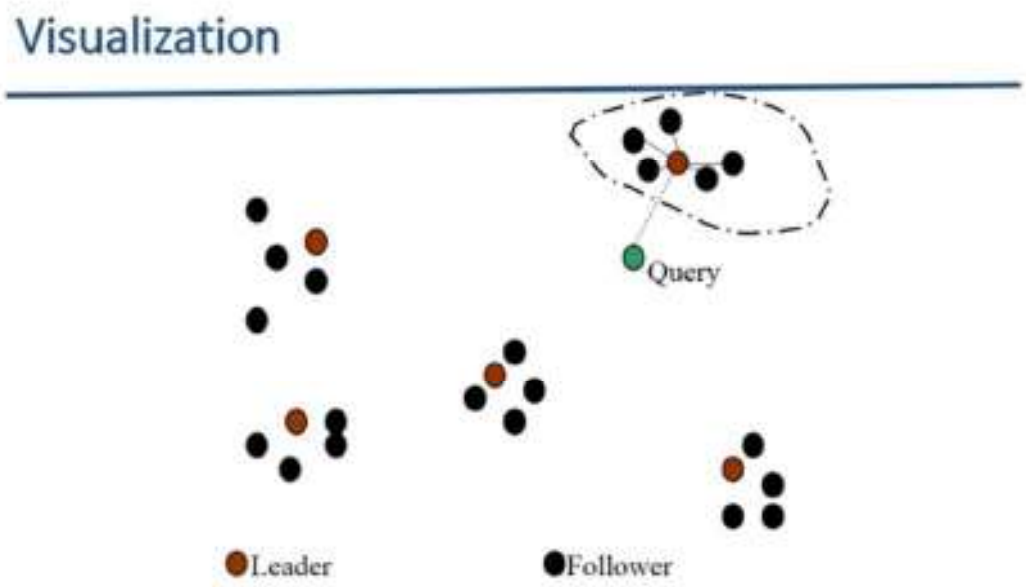
First run a pre-processing phase:

- pick $\sqrt{N}$ *docs* at random: call these *leaders*
- For each other doc, pre-compute nearest leader
    - Docs attached to a leader: its *followers;*
    - <u>Likely:</u> each leader has ~ $\sqrt{N}$ followers.

Process a query as follows:

- Given query *Q.* find its nearest *leader L.*
- Seek *k* nearest docs from among *L*'s followers.

Visualization



Why use random sampling

- Fast
- Leaders reflect data distribution

Sampling and pre-grouping

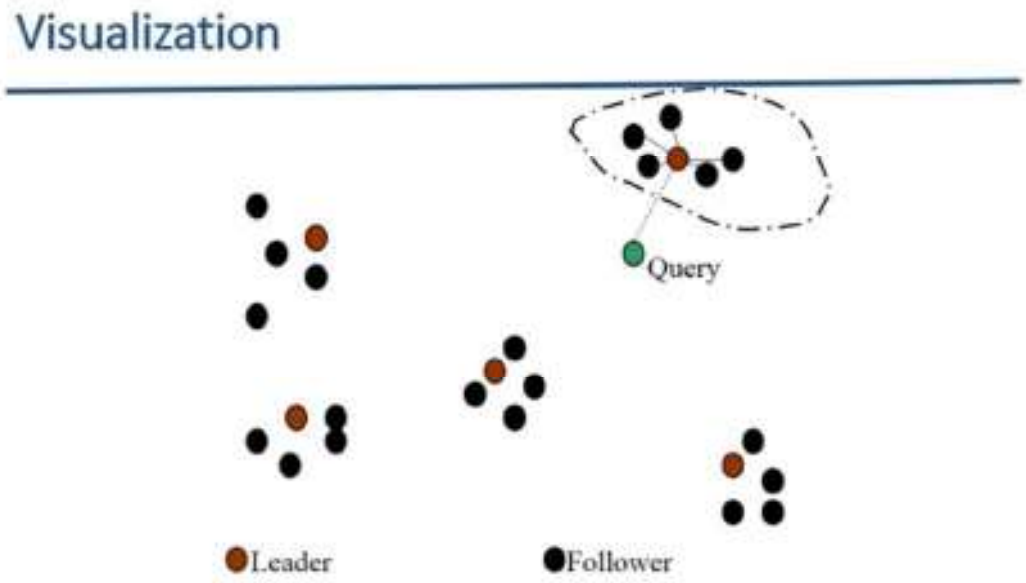
First run a pre-processing phase:

- pick $\sqrt{N}$ *docs* at random: call these *leaders*
- For each other doc, pre-compute nearest leader
    - Docs attached to a leader: its *followers;*
    - Likely: each leader has ~ $\sqrt{N}$ followers.

Process a query as follows:

- Given query *Q.* find its nearest *leader L.*
- Seek *k* nearest docs from among *L*'s followers.

Visualization



General variants

- Have each follower attached to a=3 (say) nearest leaders.
- From query, find b=4 (say) nearest leaders and their followers.
- Can recur on leader/follower construction.

Lecture # 31

- Dimensionality reduction

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Dimensionality reduction
- Random projection onto *k<<m* axes
- Computing the random projection
- Latent semantic indexing (LSI)
- The matrix
- Dimension reduction

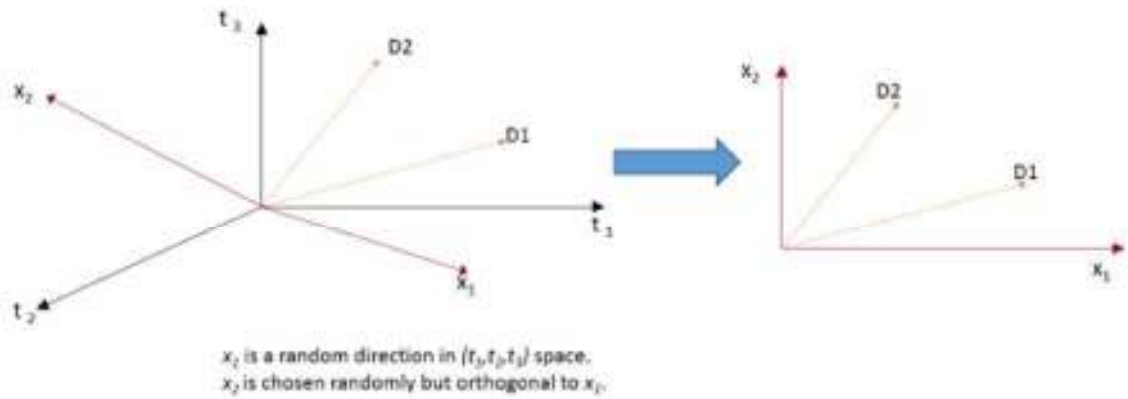
Dimensionality reduction

- What if we could take our vectors and "pack" them into fewer dimensions (say 10000→100) while preserving distances?
- (Well, almost.)
    - Speeds up cosine computations.
- Two methods:
    - Random projection.
    - "Latent semantic indexing".

Random projection onto *k<<m* axes.

- Choose a random direction x_1 in the vector space.
- For *i = 2* to *k*,
    - Choose a random direction x_i that is orthogonal to $x_1, x_2, \ldots x_{i-1}$.
- Project each doc vector into the subspace $x_1, x_2, \ldots x_k$.

E.g., from 3 to 2 dimensions



Guarantee

- With high probability, relative distances are (approximately) preserved by projection.
- Pointer to precise theorem in Resources.

Computing the random projection

- Projecting *n* vectors from *m* dimensions down to *k* dimensions:
    - Start with $m \times n$ matrix of terms $\times$ docs, *A*.
    - Find random $k \times m$ orthogonal projection matrix *R*.
    - Compute matrix product $W = R \times A$.
- *j*th column of *W* is the vector corresponding to doc *j*, but now in $k << m$ dimensions.

Cost of computation

- This takes a total of *kmn* multiplications.
- Expensive - see Resources for ways to do essentially the same thing, quicker.
- *Exercise: by projecting from 10000 dimensions down to 100, are we really going to make each cosine computation faster?*
    - *Size of the vectors would decrease*
    - *Will result into smaller postings*

Latent semantic indexing (LSI)

- Another technique for dimension reduction
- Random projection was data-*independent*
- LSI on the other hand is data-*dependent*
    - Eliminate redundant axes
    - Pull together "related" axes
        - ***car*** and ***automobile***

Notions from linear algebra

- Matrix, vector
- Matrix transpose and product
- Rank
- Eigenvalues and eigenvectors.

The matrix

- The matrix $\begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix}$

has rank 2: the first two rows are linearly independent, so the rank is at least 2, but all three rows are linearly dependent (the first is equal to the sum of the second and third) so the rank must be less than 3.

- The matrix $A = \begin{bmatrix} 1 & 1 & 0 & 2 \\ -1 & -1 & 0 & -2 \end{bmatrix}$

has rank 1: there are nonzero colums. So the rank is positive. but any pair of columns is linearly dependent.

The matrix

Similarly, the transpose $A^T = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 0 & 0 \\ 2 & -2 \end{bmatrix}$

of *A* has rank 1. Indeed, since the column vectors of *A* are the row vectors of the transpose of *A*. the statement that the column rank of a matrix equals its row rank is equivalent to the statement that the rank of a matrix is equal to the rank of its transpose, i.e., rk(A) = rk(A).

Singular-Value Decomposition

- Recall $m \times n$ matrix of terms $\times$ docs, *A*.
    - *A* has rank $r \leq m,n$.

- Define term-term correlation matrix $T=AA^t$
  - A^t denotes the matrix transpose of A.
  - T is a square, symmetric $m \times m$ matrix.
- Doc-doc correlation matrix $D=A^tA$.
  - D is a square, symmetric $n \times n$ matrix.

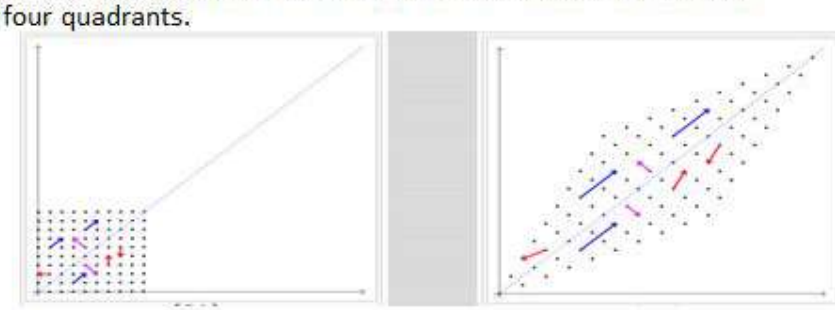
why?

Eigenvectors

- Denote by P the $m \times r$ matrix of eigenvectors of T.
- Denote by R the $n \times r$ matrix of eigenvectors of D.
- It turns out A can be expressed (decomposed) as $A = PQR^t$
  - Q is a <u>diagonal</u> matrix with the eigenvalues of AA^t in sorted order.

Eigen Vectors

- **The transformation matrix $\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ preserves the direction of**
- **vectors parallel to $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (in blue) and $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ (in violet). The**
- **points that lie on the line through the origin, parallel to an eigenvector, remain on the line after the transformation. The vectors in red are not eigenvectors, therefore their direction is altered by the transformation. See also: An** extended version. showing all four quadrants.



Eigen Vectors and Eigen Values

- $Av = \lambda v$

For the transformation matrix

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix},$$

the vector

$$v = \begin{bmatrix} 4 \\ -4 \end{bmatrix}$$

is an eigenvector with eigenvalue 2. Indeed,

$$Av = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ -4 \end{bmatrix} = \begin{bmatrix} 3 \cdot 4 + 1 \cdot (-4) \\ 1 \cdot 4 + 3 \cdot (-4) \end{bmatrix}$$
$$= \begin{bmatrix} 8 \\ -8 \end{bmatrix} = 2 \cdot \begin{bmatrix} 4 \\ -4 \end{bmatrix}.$$

On the other hand the vector

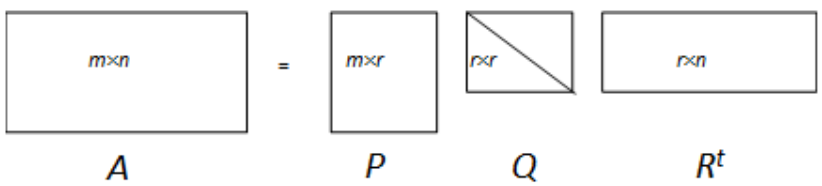
$$v = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

is *not* an eigenvector, since

$$\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \cdot 0 + 1 \cdot 1 \\ 1 \cdot 0 + 3 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix},$$

and this vector is not a multiple of the original vector v.

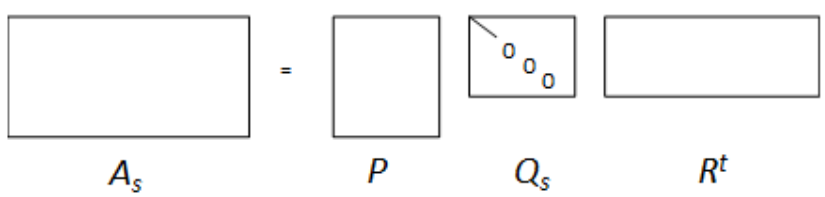
Visualization



Dimension reduction

- For some $s << r$, zero out all but the s biggest eigenvalues in Q.
  - Denote by Q_s this new version of Q.
  - Typically s in the hundreds while r could be in the (tens of) thousands.
- Let $A_s = P Q_s R^t$
- Turns out A_s is a pretty good approximation to A.

Visualization



The columns of A_s represent the docs, but in $s<<m$ dimensions.

Guarantee

- Relative distances are (approximately) preserved by projection:
    - Of all $m \times n$ rank s matrices, A_s is the best approximation to A.
- Pointer to precise theorem in Resources.

Doc-doc similarities

- $A_s A_s^t$ is a matrix of doc-doc similarities:
    - the *(j,k)* entry is a measure of the similarity of doc *j* to doc *k*.

Semi-precise intuition

- We accomplish more than dimension reduction here:
    - Docs with lots of overlapping terms stay together
    - Terms from these docs also get pulled together.
- Thus ***car*** and ***automobile*** get pulled together because both co-occur in docs with ***tires, radiator, cylinder***, etc.

Query processing

- View a query as a (short) doc:
    - call it row 0 of A_s.
- Now the entries in row 0 of $A_s A_s^t$ give the similarities of the query with each doc.
- Entry *(0,j)* is the score of doc *j* on the query.
- *Exercise*: *fill in the details of scoring/ranking.*
- LSI is expensive in terms of computation…
- Randomly choosing a subset of documents for dimensional reduction can give a significant boost in performance.

Resources

- Random projection theorem: http://citeseer.nj.nec.com/dasgupta99elementary.html
- Faster random projection: http://citeseer.nj.nec.com/frieze98fast.html
- Latent semantic indexing: http://citeseer.nj.nec.com/deerwester90indexing.html
- Books: MG 4.6, MIR 2.7.2.

Lecture # 32

- Web Search

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- The World Wide Web
- Web Pre-History
- Web Search History
- Web Challenges for IR
- Graph Structure in the Web (Bowtie)
- Zipf's Law on the Web
- Manual Hierarchical Web Taxonomies
- Business Models for Web Search

The World Wide Web

- Developed by Tim Berners-Lee in 1990 at CERN to organize research documents available on the Internet.
- Combined idea of documents available by FTP with the idea of *hypertext* to link documents.
- Developed initial HTTP network protocol, URLs, HTML, and first "web server."

Web Pre-History

- Ted Nelson developed idea of hypertext in 1965.
- Doug Engelbart invented the mouse and built the first implementation of hypertext in the late 1960's at SRI.
- ARPANET was developed in the early 1970's.
- The basic technology was in place in the 1970's; but it took the PC revolution and widespread networking to inspire the web and make it practical.

Web Browser History

- Early browsers were developed in 1992 (Erwise, ViolaWWW).
- In 1993, Marc Andreessen and Eric Bina at UIUC NCSA developed the Mosaic browser and distributed it widely.

- Andreessen joined with James Clark (Stanford Prof. and Silicon Graphics founder) to form Mosaic Communications Inc. in 1994 (which became Netscape to avoid conflict with UIUC).
- Microsoft licensed the original Mosaic from UIUC and used it to build Internet Explorer in 1995.

Search Engine Early History

- By late 1980's many files were available by anonymous FTP.
- In 1990, Alan Emtage of McGill Univ. developed Archie (short for "archives")
    - Assembled lists of files available on many FTP servers.
    - Allowed regex search of these file names.
- In 1993, Veronica and Jughead were developed to search names of text files available through Gopher servers.

Web Search History

- In 1993, early web robots (spiders) were built to collect URL's:
    - Wanderer
    - ALIWEB (Archie-Like Index of the WEB)
    - WWW Worm (indexed URL's and titles for regex search)
- In 1994, Stanford grad students David Filo and Jerry Yang started manually collecting popular web sites into a topical hierarchy called Yahoo.

Web Search History (cont)

- In early 1994, Brian Pinkerton developed WebCrawler as a class project at U Wash. (eventually became part of Excite and AOL).
- A few months later, Fuzzy Maudlin, a grad student at CMU developed Lycos. First to use a standard IR system as developed for the DARPA Tipster project. First to index a large set of pages.
- In late 1995, DEC developed Altavista. Used a large farm of Alpha machines to quickly process large numbers of queries. Supported boolean operators, phrases, and "reverse pointer" queries.

Web Search Recent History

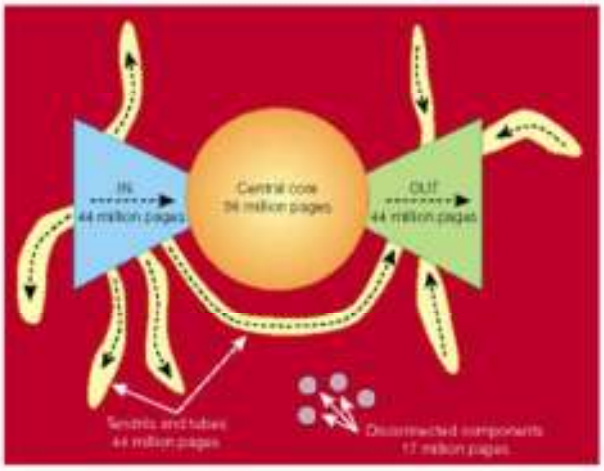
- In 1998, Larry Page and Sergey Brin, Ph.D. students at Stanford, started Google. Main advance is use of *link analysis* to rank results partially based on authority.

Web Challenges for IR

- Distributed Data: Documents spread over millions of different web servers.
- Volatile Data: Many documents change or disappear rapidly (e.g. dead links).

- Large Volume: Billions of separate documents.
- Unstructured and Redundant Data: No uniform structure, HTML errors, up to 30% (near) duplicate documents.
- Quality of Data: No editorial control, false information, poor quality writing, typos, etc.
- Heterogeneous Data: Multiple media types (images, video, VRML), languages, character sets, etc.

Graph Structure in the Web (Bowtie)



http://www9.org/w9cdrom/160/160.html

Zipf's Law on the Web

- Number of in-links/out-links to/from a page has a Zipfian distribution.
- Length of web pages has a Zipfian distribution.
- Number of hits to a web page has a Zipfian distribution.

Manual Hierarchical Web Taxonomies

- Yahoo approach of using human editors to assemble a large hierarchically structured directory of web pages.
    - http://www.yahoo.com/
- Open Directory Project is a similar approach based on the distributed labor of volunteer editors ("net-citizens provide the collective brain"). Used by most other search engines. Started by Netscape.
    - http://www.dmoz.org/

Business Models for Web Search

- Advertisers pay for banner ads on the site that do not depend on a user's query.
- Goto/Overture
    - CPM: Cost Per Mille (thousand impressions). Pay for each ad display.
    - CPC: Cost Per Click. Pay only when user clicks on ad.
    - CTR: Click Through Rate. Fraction of ad impressions that result in clicks throughs. CPC = CPM / (CTR * 1000)
    - CPA: Cost Per Action (Acquisition). Pay only when user actually makes a purchase on target site.
- Advertisers bid for "keywords". Ads for highest bidders displayed when user query contains a purchased keyword.
    - PPC: Pay Per Click. CPC for bid word ads (e.g. Google AdWords).

Lecture # 33

- Spidering

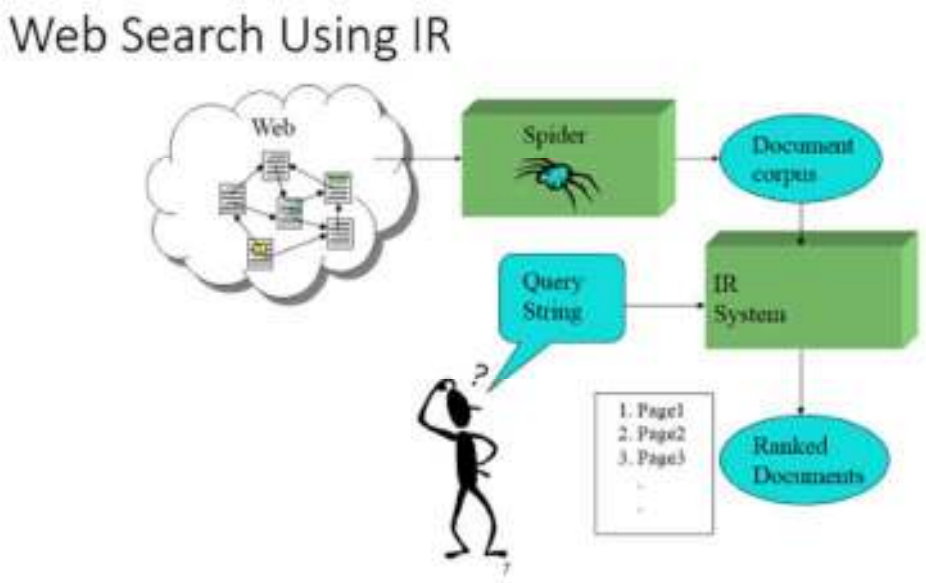
ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

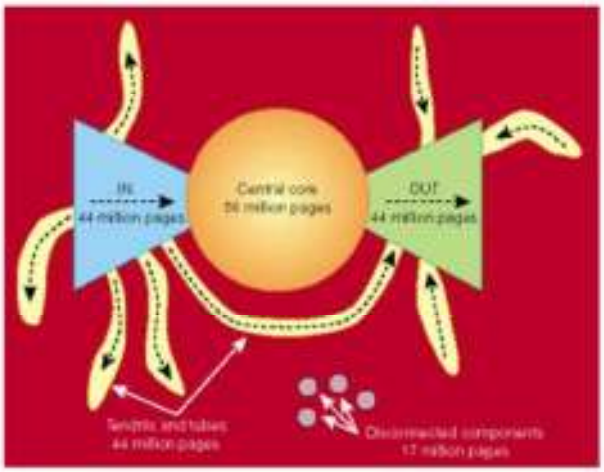
- Web Search Using IR
- Spiders (Robots/Bots/Crawlers)
- What any crawler *must* do
- What any crawler *should* do
- Search Strategies
- Basic crawl architecture
- Restricting Spidering
- Robot Exclusion



Spiders (Robots/Bots/Crawlers)

- Start with a comprehensive set of root URL's from which to start the search.
- Follow all links on these pages recursively to find additional pages.
- Index all **novel** found pages in an inverted index as they are encountered.
- May allow users to directly submit pages to be indexed (and crawled from).

Graph Structure in the Web (Bowtie)



http://www9.org/w9cdrom/160/160.html

What any crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect *robots.txt* (more on this shortly)
- Be Robust: Be immune to spider traps and other malicious behavior from web servers

What any crawler *should* do

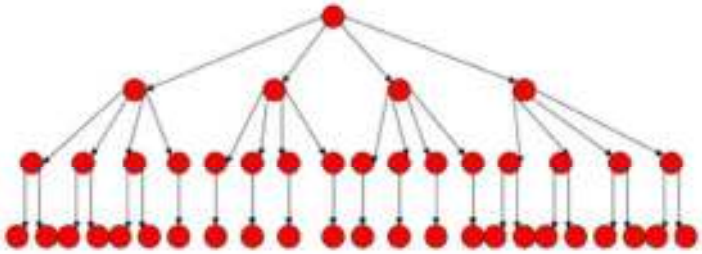
- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources

What any crawler *should* do

- Fetch pages of "higher quality" first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols

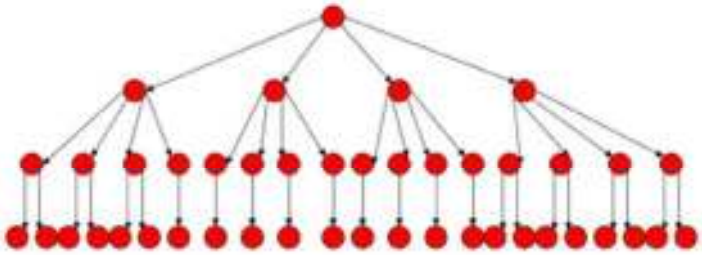
Search Strategies

Breadth-first Search



Search Strategies (cont)

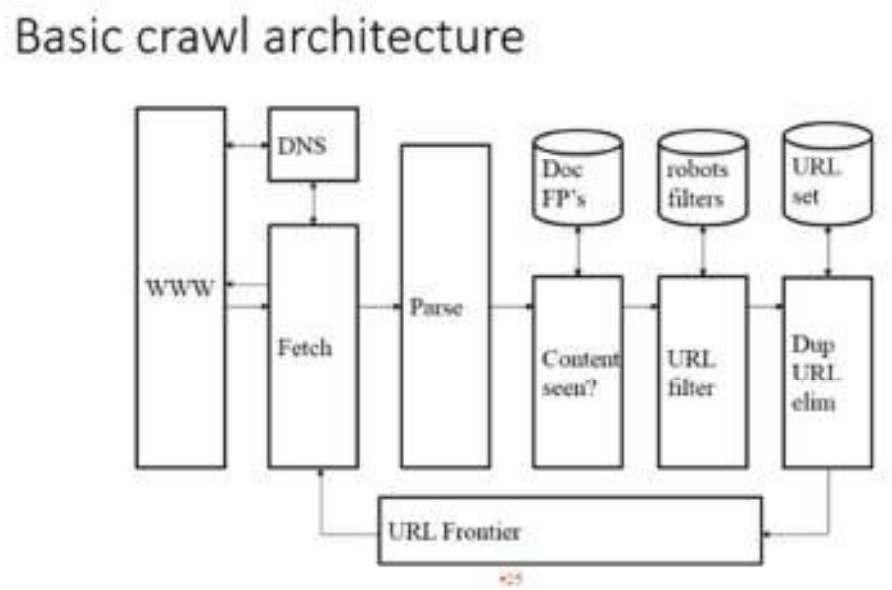
Depth-first Search



Search Strategy Trade-Off's

- Breadth-first explores uniformly outward from the root page but requires memory of all nodes on the previous level (exponential in depth). Standard spidering method.
- Depth-first requires memory of only depth times branching-factor (linear in depth) but gets "lost" pursuing a single thread.

- Both strategies implementable using a queue of links (URL's).



Avoiding Page Duplication

- Must detect when revisiting a page that has already been spidered (web is a graph not a tree).
- Must efficiently index visited pages to allow rapid recognition test.
    - Tree indexing (e.g. trie)
    - Hashtable
- Index page using URL as a key.
    - Must canonicalize URL's (e.g. delete ending "/")
    - Not detect duplicated or mirrored pages.
- Index page using textual content as a key.
    - Requires first downloading page.

Spidering Algorithm

```
Initialize queue (Q) with initial set of known URL's.
Until Q empty or page or time limit exhausted:
    Pop URL, L, from front of Q.
    If L is not to an HTML page (.gif, .jpeg, .ps, .pdf, .ppt…)
        continue loop.
    If already visited L, continue loop.
    Download page, P, for L.
    If cannot download P (e.g. 404 error, robot excluded)
        continue loop.
```

Index P (e.g. add to inverted index or store cached copy).
Parse P to obtain list of new links N.
Append N to the end of Q.

Queueing Strategy

- How new links added to the queue determines search strategy.
- FIFO (append to end of Q) gives breadth-first search.
- LIFO (add to front of Q) gives depth-first search.
- Heuristically ordering the Q gives a "focused crawler" that directs its search towards "interesting" pages.

Restricting Spidering

- Restrict spider to a particular site.
    - Remove links to other sites from Q.
- Restrict spider to a particular directory.
    - Remove links not in the specified directory.
- Obey page-owner restrictions (robot exclusion).

Link Extraction

- Must find all links in a page and extract URLs.
    - <a href="http://www.cs.utexas.edu/users/mooney/ir-course">
    - <frame src="site-index.html">
- Must complete relative URL's using current page URL:
    - <a href="proj3"> to http://www.cs.utexas.edu/users/mooney/ir-course/proj3
    - <a href="../cs343/syllabus.html"> to http://www.cs.utexas.edu/users/mooney/cs343/syllabus.html

URL Syntax

- A URL has the following syntax:
    - <scheme>://<authority><path>?<query>#<fragment>
- An *authority* has the syntax:
    - <host>:<port-number>
- A *query* passes variable values from an HTML form and has the syntax:
    - <variable>=<value>&<variable>=<value>…
- A *fragment* is also called a *reference* or a *ref* and is a pointer within the document to a point specified by an anchor tag of the form:
    - <A NAME="<fragment>">

Robot Exclusion

- Web sites and pages can specify that robots should not crawl/index certain areas.
- Two components:

- Robots Exclusion Protocol: Site wide specification of excluded directories.
- Robots META Tag: Individual document tag to exclude indexing or following links.

Robots Exclusion Protocol

- Site administrator puts a "robots.txt" file at the root of the host's web directory.
  - http://www.ebay.com/robots.txt
  - http://www.cnn.com/robots.txt
- File is a list of excluded directories for a given robot (user-agent).
  - Exclude all robots from the entire site:

```
User-agent: *
 Disallow: /
```

Robot Exclusion Protocol Examples

- Exclude specific directories:

```
User-agent: *
 Disallow: /tmp/
 Disallow: /cgi-bin/
 Disallow: /users/paranoid/
```

- Exclude a specific robot:

```
User-agent: GoogleBot
 Disallow: /
```

- Allow a specific robot:

```
User-agent: GoogleBot
 Disallow:
 User-agent: *
 Disallow: /
```

Robots META Tag

- Include META tag in HEAD section of a specific HTML document.
  - <meta name="robots" content="none">
- Content value is a pair of values for two aspects:
  - index | noindex: Allow/disallow indexing of this page.
  - follow | nofollow: Allow/disallow following links on this page.

Robots META Tag (cont)

- Special values:
    - all = index,follow
    - none = noindex,nofollow
- Examples:

<meta name="robots" content="noindex,follow">

<meta name="robots" content="index,nofollow">

<meta name="robots" content="none">

Lecture # 34

- Web Crawler

ACKNOWLEDGEMENTS

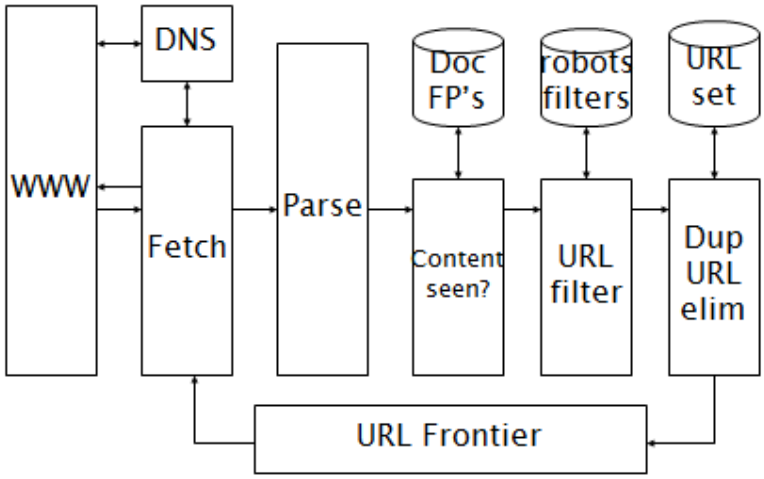
The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

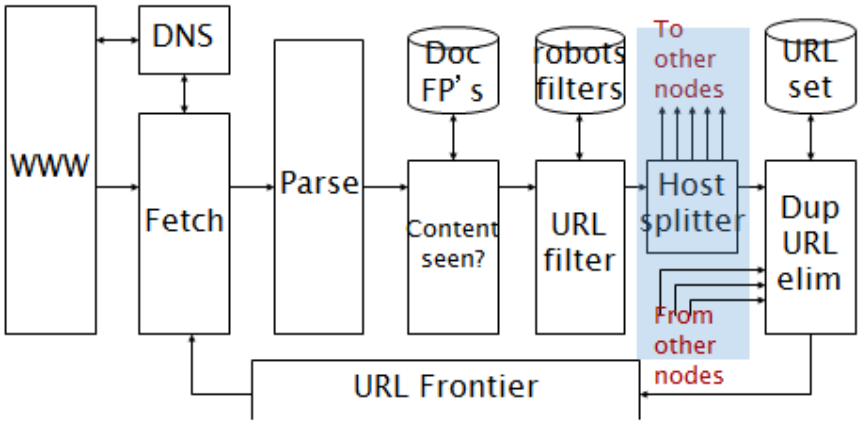
- Basic crawl architecture
- Communication between nodes
- URL frontier: Mercator scheme
- Duplicate documents
- Shingles + Set Intersection

Basic crawl architecture



Communication between nodes

- Output of the URL filter at each node is sent to the Dup URL Eliminator of the appropriate node



URL frontier: two main considerations

- Politeness: do not hit a web server too frequently
- Freshness: crawl some pages more often than others
    - E.g., pages (such as News sites) whose content changes often

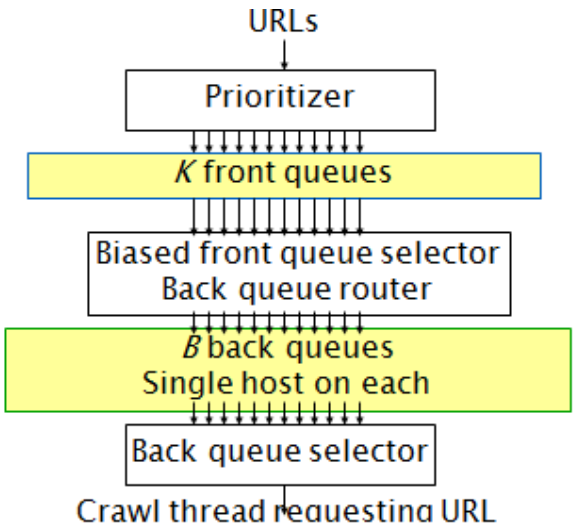
These goals may conflict each other.

(E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

Politeness – challenges

- Even if we restrict only one thread to fetch from a host, can hit it repeatedly
- Common heuristic: insert time gap between successive requests to a host that is >> time for most recent fetch from that host

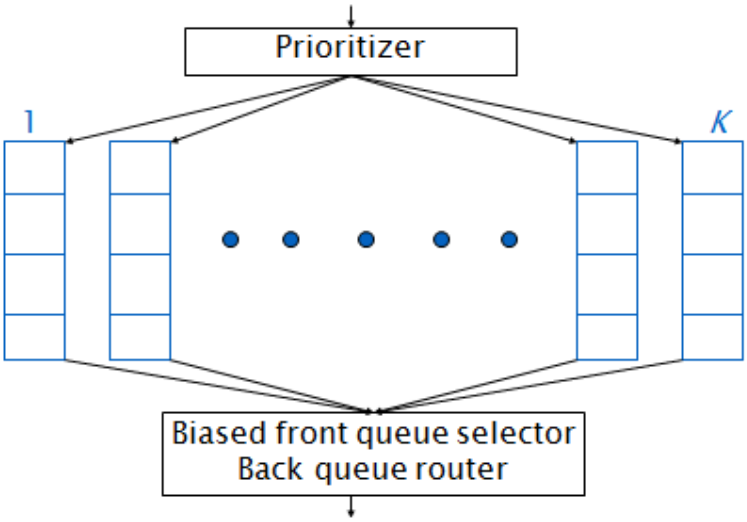
URL frontier: Mercator scheme



Mercator URL frontier

- URLs flow in from the top into the frontier
- Front queues manage prioritization
- Back queues enforce politeness
- Each queue is FIFO

Front queues



Front queues

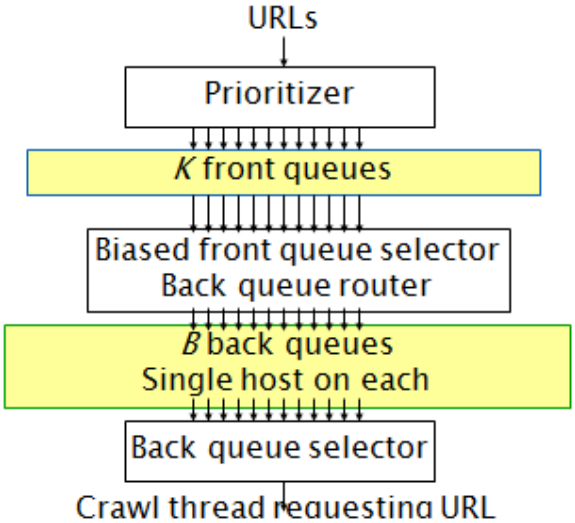
- Prioritizer assigns to URL an integer priority between 1 and *K*
  - Appends URL to corresponding queue
- Heuristics for assigning priority
  - Refresh rate sampled from previous crawls
  - Application-specific (e.g., "crawl news sites more often")

Back queue invariants

- Each back queue is kept non-empty while the crawl is in progress
- Each back queue only contains URLs from a single host
  - Maintain a table from hosts to back queues

Host name	Back queue
…	3
	1
	B

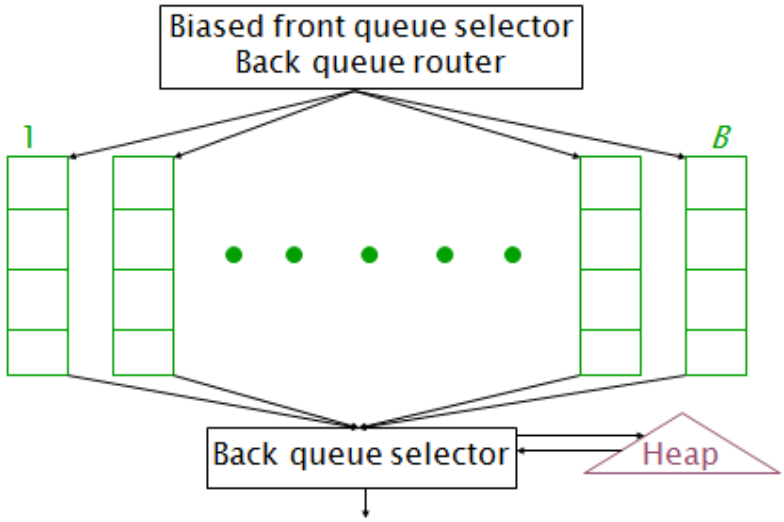
URL frontier: Mercator scheme



Biased front queue selector

- When a back queue requests a URL (in a sequence to be described): picks a front queue from which to pull a URL
- This choice can be round robin biased to queues of higher priority, or some more sophisticated variant
  - Can be randomized

Back queues



Back queue heap

- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again
- This earliest time is determined from
    - Last access to that host
    - Any time buffer heuristic we choose

Back queue processing

- A crawler thread seeking a URL to crawl:
- Extracts the root of the heap
- Fetches URL at head of corresponding back queue *q* (look up from table)
- Checks if queue *q* is now empty – if so, pulls a URL *v* from front queues
    - If there's already a back queue for *v*'s host, append *v* to *q* and pull another URL from front queues, repeat
    - Else add *v* to *q*
- When *q* is non-empty, create heap entry for it

Number of back queues *B*

- Keep all threads busy while respecting politeness
- Mercator recommendation: three times as many back queues as crawler threads

Duplicate documents

- The web is full of duplicated content
- Strict duplicate detection = exact match
    - Not as common
- But many, many cases of near duplicates
    - E.g., Last modified date the only difference between two copies of a page

Duplicate/Near-Duplicate Detection

- *Duplication*: Exact match can be detected with fingerprints
- *Near-Duplication*: Approximate match
    - Overview
        - Compute syntactic similarity with an edit-distance measure
        - Use similarity threshold to detect near-duplicates
            - E.g., Similarity > 80% => Documents are "near duplicates"
            - Not transitive though sometimes used transitively

Computing Similarity

- Features:
    - Segments of a document (natural or artificial breakpoints)
    - Shingles (Word N-Grams)
    - ***a rose is a rose is a rose*** → 4-grams are

a_rose_is_a

rose_is_a_rose

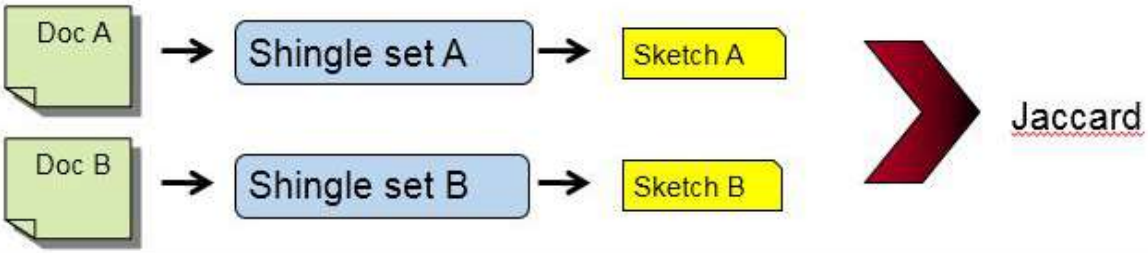
is_a_rose_is

a_rose_is_a

- Similarity Measure between two docs (= sets of shingles)
    - Set intersection
    - Specifically (Size_of_Intersection / Size_of_Union)

Shingles + Set Intersection

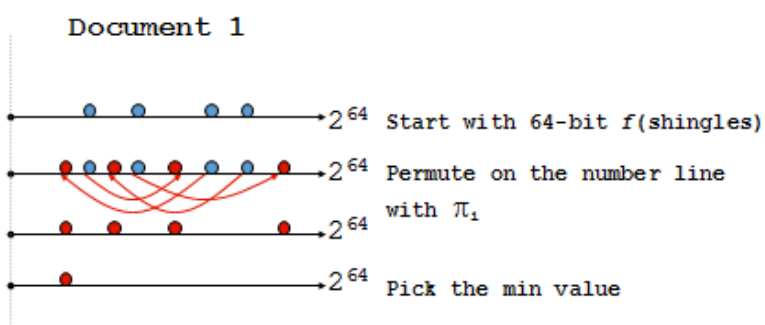
- Computing exact set intersection of shingles between all pairs of documents is expensive/intractable
    - Approximate using a cleverly chosen subset of shingles from each (a *sketch*)
- Estimate (size_of_intersection / size_of_union) based on a short sketch



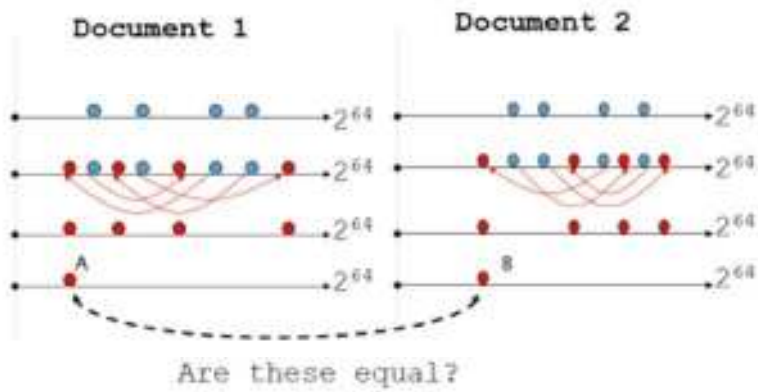
Sketch of a document

- Create a "sketch vector" (of size ~200) for each document
  - Documents that share ≥ *t* (say 80%) corresponding vector elements are deemed near duplicates
  - For doc *D*, sketch$_D$[*i*] is as follows:
    - Let f map all shingles in the universe to $0..2^m$ (e.g., f = fingerprinting)
    - Let π_i be a *random permutation* on $0..2^m$
    - Pick MIN $\{\pi_i(f(s))\}$ over all shingles *s* in *D*

Computing Sketch[i] for Doc1

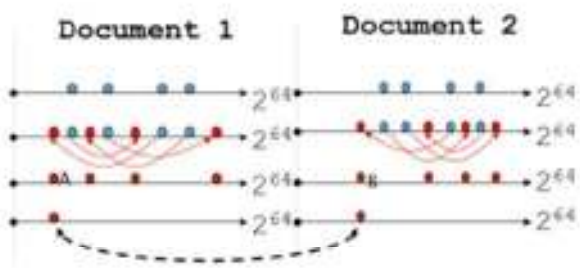


Test if Doc1.Sketch[i] = Doc2.Sketch[i]



Test for 200 random permutations: $\pi_1, \pi_2, \ldots \pi_{200}$

However…



A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (i.e., lies in the intersection)

Claim: This happens with probability
Size_of_intersection / Size_of_union

Final notes

- Shingling is a *randomized algorithm*
  - Our analysis did not presume any probability model on the inputs
  - It will give us the right (wrong) answer with some probability on *any input*

- We've described how to detect near duplication in a pair of documents
- In "real life" we'll have to concurrently look at many pairs
    - Use Locality Sensitive Hashing for this

Lecture # 35

- Distributed Index

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the underline sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Distributed Index
- Map Reduce
- Mapping onto Hadoop Map Reduce
- Dynamic Indexing
- Issues in Dynamic Indexing

Introduction

- Inverted Index in Memory
- Inverted Index on a HD of a machine
- Inverted Index on a number of machines
    - Cluster Computing/Grid Computing
    - Data Centers
    - Large scale IR systems have number of Data Centers.

Map Reduce

- Utilize several machines to construct
- Map Phase
    - Take documents and parse them
    - Split a-h; i-p; q-z
- Reduce Phase
    - Inversion → Build Inverted index out of it.
- Apache Hadoop is an open source frame work.
- Each phase of an IR system can be mapped to map-reduce

Mapping onto Hadoop Map Reduce

- Map takes documents and comes up with
    - TermId, docId
- Reducer takes [TermId, DocId] pairs
    - Based on our defined mapping criterion e.g docIds or a-g etc these [termId,docId] pairs are assigned to different reducers.
    - Which further sort the information based on docId and generate postings lists for different terms.

Dynamic Indexing

- Documents are updated, deleted
- Simple Approach
    - Maintain a big main index.
    - New documents in auxiliary small index to be built in say RAM.
    - Answer queries by involving both indexes
    - **For Deletions**
        - Invalidation bit vectors for deleted docs
        - Size of the bit-vector is equal to the size of docs in corpus
        - Filter the search results by using the bit-vector
    - Periodically, merge the main and aux indexes.
    - Keep it simple and assume index is residing on a single machine

Dynamic Indexing (Cont….)

- **Merge operation is expensive (raises performance issues)**
    - Merging may be efficient if we maintain a separate file for each posting → Merging would only be an append operation like this.
    - But this will result into a large number of files, and this makes the OS slow
    - We need to read each file while merging.

Dynamic Indexing (Logarithmic Merge)

- Logarithmic Merge
    - Each index is double than the previous one.
    - Z_0 index in memory
    - I1 , I2, I3… in memory
    - At a given time some of them will exist.
    - Z_0 gets bigger than a certain value 'n' the number of tokens in that index, merge it with I1.
    - Z_0 again get bigger enough again merge it with I1, now there exist I1.
    - Z_0 again get big merge it with I1
    - Search using all indexes
    - If there are T terms in the index, then we shall have at most log(T) indexes.
    - Now query processing requires O(logT) merges.
    - There will be less than logT indexes in total

Issues in Dynamic Indexing

- **Collection Statistics** → IDF of a term is harder to maintain in the presence of multiple indexes.
- **Spelling correction** that we discussed assumed that there is just a single index.
- **Invalidation bit-vectors** are supposed to be incorporated while computing the statistics of the words

- **Solutions**
- A simple heuristic is to **utilize LARGEST** index, but it reflects inaccurate information.
- **The large search engines keep on building new index on a separate set of machines**, and once it is constructed they start using it.
    - Google Dance
    - Think about expanding it while incorporating positional index.

Lecture # 36

- Link Analysis

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

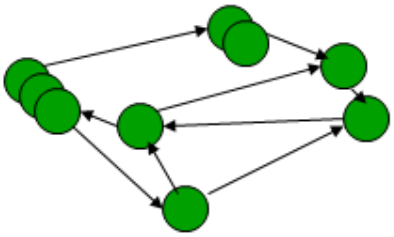
1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Hypertext and Links
- The Web as a Directed Graph
- Anchor Text
- Indexing anchor text
- PageRank scoring

Hypertext and Links

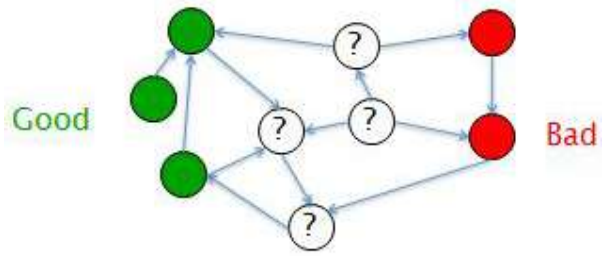
- We look beyond the *content* of documents
    - We begin to look at the hyperlinks between them
- Address questions like
    - Do the links represent a conferral of authority to some pages? Is this useful for ranking?
    - How likely is it that a page pointed to by the CERN home page is about high energy physics
- Big application areas
    - The Web
    - Email
    - Social networks



Links are everywhere

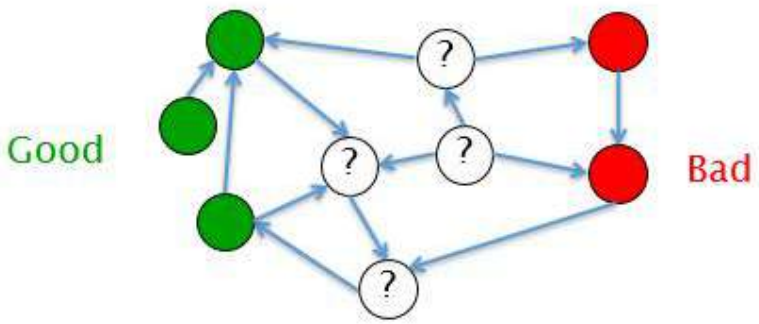
- Powerful sources of authenticity and authority
    - Mail spam – which email accounts are spammers?
    - Host quality – which hosts are "bad"?
    - Phone call logs

- The Good, The Bad and The Unknown



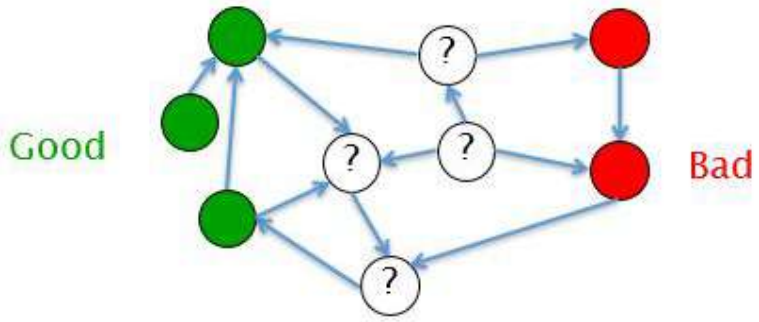
Example 1: Good/Bad/Unknown

- The Good, The Bad and The Unknown
  - Good nodes won't point to Bad nodes
  - All other combinations plausible



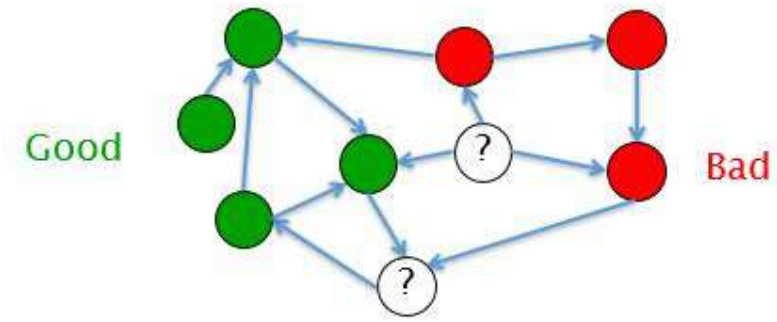
Simple iterative logic

- Good nodes won't point to Bad nodes
  - If you point to a Bad node, you're Bad
  - If a Good node points to you, you're Good



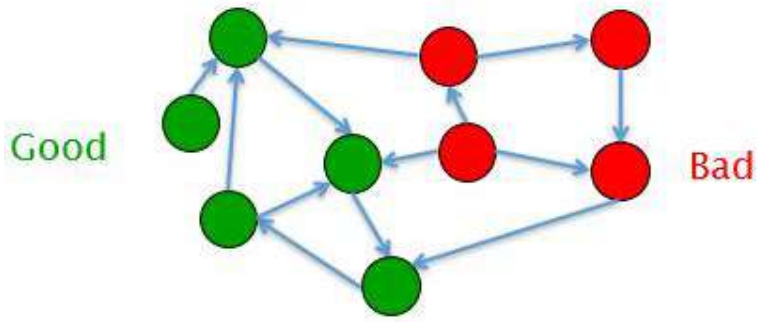
Simple iterative logic

- Good nodes won't point to Bad nodes
    - If you point to a Bad node, you're Bad
    - If a Good node points to you, you're Good



Simple iterative logic

- Good nodes won't point to Bad nodes
    - If you point to a Bad node, you're Bad
    - If a Good node points to you, you're Good

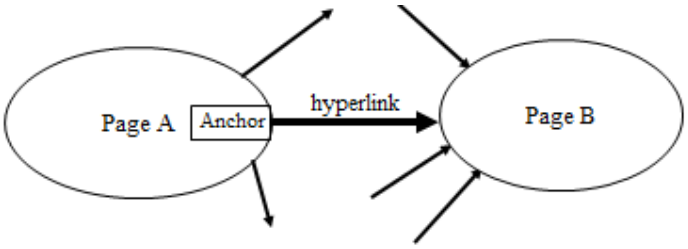


Sometimes need probabilistic analogs – e.g., mail spam

Our primary interest in this course

- Link analysis for most IR functionality thus far based purely on text
    - Scoring and ranking
    - Link-based clustering – topical structure from links
    - Links as features in classification – documents that link to one another are likely to be on the same subject
- Crawling
    - Based on the links seen, where do we crawl next?

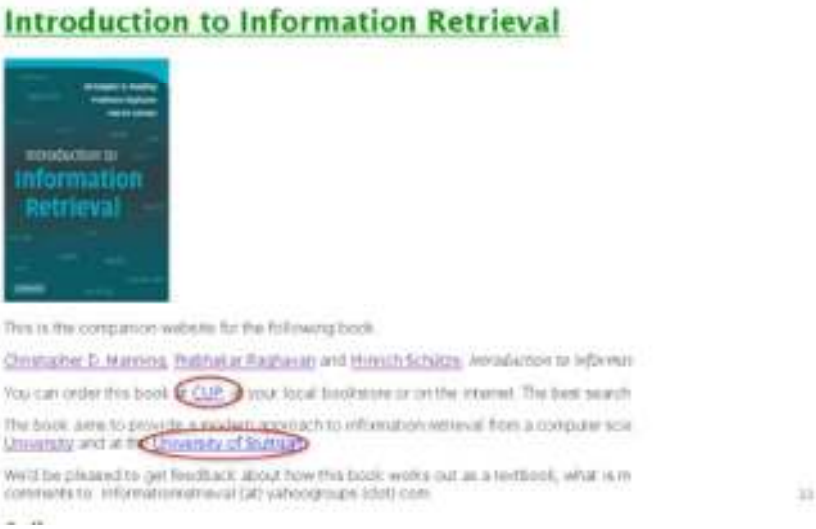
The Web as a Directed Graph



Hypothesis 1: A hyperlink between pages denotes a conferral of authority (quality signal)

Hypothesis 2: The text in the anchor of the hyperlink on page A describes the target page B

Assumption 1: reputed sites

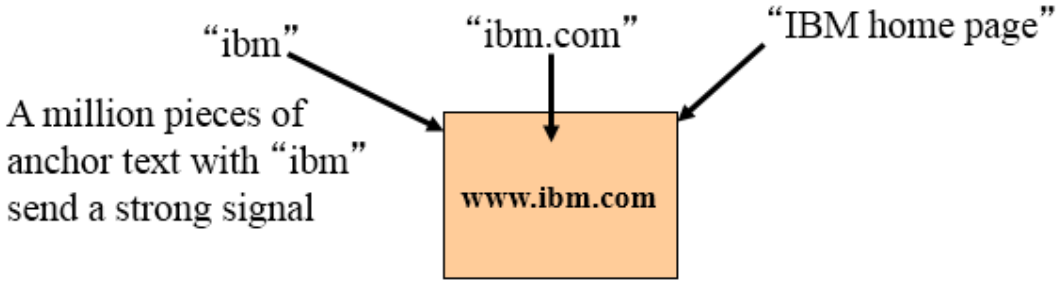


Assumption 2: annotation of target

Anchor Text

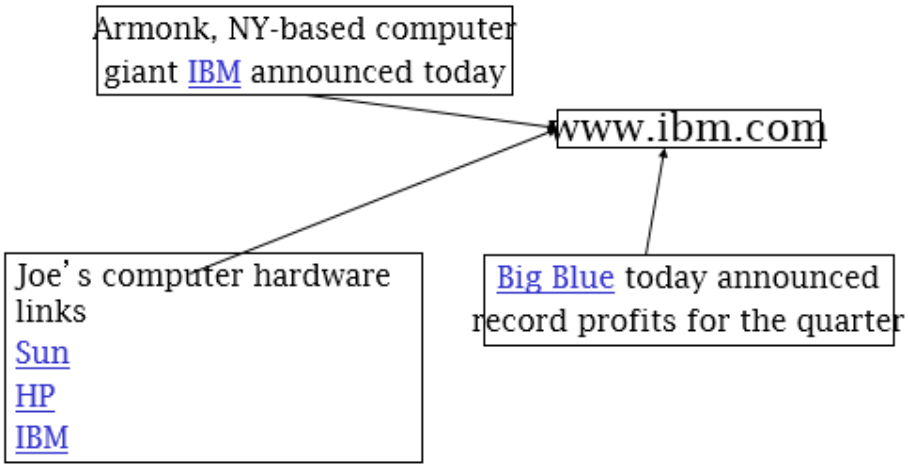
WWW Worm- McBryan [Mcbr94]

- For ***ibm*** how to distinguish between:
    - IBM's home page (mostly graphical)
    - IBM's copyright page (high term freq. for 'ibm')
    - Rival's spam page (arbitrarily high term freq.)



Indexing anchor text

- When indexing a document *D*, include (with some weight) anchor text from links pointing to *D*.



Adjacency lists

- The set of neighbors of a node
- Assume each URL represented by an integer
- E.g., for a 4 billion page web, need 32 bits per node
- Naively, this demands 64 bits to represent each hyperlink
- Boldi/Vigna get down to an average of ~3 bits/link
    - Further work achieves 2 bits/link

Main ideas of Boldi/Vigna

- Consider lexicographically ordered list of all URLs, e.g.,
    - www.stanford.edu/alchemy
    - www.stanford.edu/biology
    - www.stanford.edu/biology/plant
    - www.stanford.edu/biology/plant/copyright
    - www.stanford.edu/biology/plant/people
    - www.stanford.edu/chemistry

Boldi/Vigna

- Each of these URLs has an adjacency list
- Main idea: due to templates, the adjacency list of a node is similar to one of the 7 preceding URLs in the lexicographic ordering
- Express adjacency list in terms of one of these
- E.g., consider these adjacency lists
  - 1, 2, 4, 8, 16, 32, 64
  - 1, 4, 9, 16, 25, 36, 49, 64
  - 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
  - 1, 4, 8, 16, 25, 36, 49, 64

Why 7?

Encode as (−2), remove 9, add 8

Gap encodings

- Given a sorted list of integers x, y, z, …, represent by x, y-x, z-y, …
- Compress each integer using a code
  - γ code - Number of bits = $1 + 2\lfloor \lg x \rfloor$
  - δ code: …
  - Information theoretic bound: $1 + \lfloor \lg x \rfloor$ bits
  - ζ code: Works well for integers from a power law Boldi Vigna DCC 2004

Citation Analysis

- Citation frequency
- Bibliographic coupling frequency
  - Articles that co-cite the same articles are related
- Citation indexing
  - Who is this author cited by? (Garfield 1972)
- Pagerank preview: Pinsker and Narin '60s
  - Asked: which journals are authoritative?

The web isn't scholarly citation

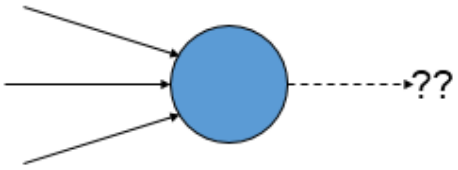
- Millions of participants, each with self interests
- Spamming is widespread
- Once search engines began to use links for ranking (roughly 1998), link spam grew
  - You can join a *link farm* – a group of websites that heavily link to one another

PageRank scoring

- Imagine a browser doing a random walk on web pages:
    - Start at a random page
    - At each step, go out of the current page along one of the links on that page, equiprobably
- "In the long run" each page has a long-term visit rate - use this as the page's score.

Not quite enough

- The web is full of dead-ends.
    - Random walk can get stuck in dead-ends.
    - Makes no sense to talk about long-term visit rates.



Teleporting

- At a dead end, jump to a random web page.
- At any non-dead end, with probability 10%, jump to a random web page.
    - With remaining probability (90%), go out on a random link.
    - 10% - a parameter.

Result of teleporting

- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited (not obvious, will show this).
- How do we compute this visit rate?

Resources

- IIR Chap 21
- http://www2004.org/proceedings/docs/1p309.pdf
- http://www2004.org/proceedings/docs/1p595.pdf
- http://www2003.org/cdrom/papers/refereed/p270/kamvar-270-xhtml/index.html
- http://www2003.org/cdrom/papers/refereed/p641/xhtml/p641-mccurley.html
- The WebGraph framework I: Compression techniques (Boldi et al. 2004)

Lecture # 37

- Markov chains

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

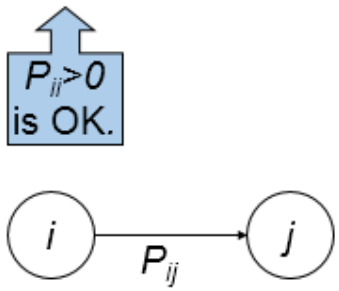
1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Markov chains
- Ergodic Markov chains
- Markov Chain with Teleporting
- Query Processing
- Personalized PageRank

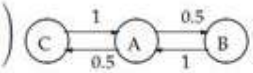
Markov chains

- A Markov chain consists of *n* states, plus an *n*×*n* transition probability matrix **P**.
- At each step, we are in one of the states.
- For $1 \le i,j \le n$, the matrix entry P_{ij} tells us the probability of *j* being the next state, given we are currently in state *i*.



Markov chains

$$\forall i, j, P_{ij} \in [0,1] \quad \begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$



- Clearly, for all i, $\sum^{n} P_{ij} = 1$.

A matrix with non-negative entries that satisfies Equation (21.1) is known as a *stochastic matrix*. A key property of a stochastic matrix is that it has a *principal left eigenvector* corresponding to its largest eigenvalue, which is 1.

- Markov chains are abstractions of random walks.
- A simple markov chain with three nodes is shown below. The numbers on links show the transition probabilities.
- **In markov chain, the prob. Distribution of the next step only depends on the current state, and not on how markov chain arrived at current state.**

Ergodic Markov chains

- For any *ergodic* Markov chain, there is a unique long-term visit rate for each state.
  - *Steady-state probability distribution*.
- Over a long time-period, we visit each state in proportion to this rate.
- It does not matter where we start.

Definition: A Markov chain is said to be *ergodic* if there exists a positive integer T_0 such that for all pairs of states i, j in the Markov chain, if it is started at time 0 in state i then for all $t > T_0$, the probability of being in state j at time t is greater than 0.

For a Markov chain to be ergodic, two technical conditions are required of its states and the non-zero transition probabilities; these conditions are known as *irreducibility* and *aperiodicity*. Informally, the first ensures that there is a sequence of transitions of non-zero probability from any state to any other, while the latter ensures that the states are not partitioned into sets such that all state transitions occur cyclically from one set to another.

Markov Chain with Teleporting: Random Surfer Model

1. If a row of A has no 1's, then replace each element by $1/N$. For all other rows proceed as follows.
2. Divide each 1 in A by the number of 1's in its row. Thus, if there is a row with three 1's, then each of them is replaced by $1/3$.
3. Multiply the resulting matrix by $1 - \alpha$.
4. Add α/N to every entry of the resulting matrix, to obtain P.

We consider the web graph in Exercise 21.6 with $\alpha = 0.5$. The transition probability matrix of the surfer's walk with teleportation is then

	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

$$P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix}.$$

Imagine that the surfer starts in state 1, corresponding to the initial probability distribution vector $\vec{x}_0 = (1\ 0\ 0)$. Then, after one step the distribution is

$$\vec{x}_0 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} = \vec{x}_1.$$

$$\vec{x}_1 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix} = \vec{x}_2.$$

$\vec{x}_0$	1	0	0
$\vec{x}_1$	1/6	2/3	1/6
$\vec{x}_2$	1/3	1/3	1/3
$\vec{x}_3$	1/4	1/2	1/4
$\vec{x}_4$	7/24	5/12	7/24
...	...	...	...
$\vec{x}$	5/18	4/9	5/18

Query Processing

- Compute Page Rank for all pages
- Run the query
- Sort the obtained pages based on the page rank
- Integrate page rank and relevance to come up with the final result

Personalized PageRank

- PageRank can be biased (personalized) by changing **E** to a non-uniform distribution.
- Restrict "random jumps" to a set of specified relevant pages.
- For example, let $E(p) = 0$ except for one's own home page, for which $E(p) = \alpha$
- This results in a bias towards pages that are closer in the web graph to your own homepage.
- 0.6*sports + 0.4*news

Google PageRank-Biased Spidering

- Use PageRank to direct (focus) a spider on "important" pages.
- Compute page-rank using the current set of crawled pages.
- Order the spider's search queue based on current estimated PageRank.
- Topic Directed Page Rank

Resources

- IIR Chap 21
- http://www2004.org/proceedings/docs/1p309.pdf
- http://www2004.org/proceedings/docs/1p595.pdf
- http://www2003.org/cdrom/papers/refereed/p270/kamvar-270-xhtml/index.html
- http://www2003.org/cdrom/papers/refereed/p641/xhtml/p641-mccurley.html
- The WebGraph framework I: Compression techniques (Boldi et al. 2004)

Lecture # 38

- HITS

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Hyper-link induced topic search (HITS)
- Hubs and Authorities
- The hope
- Distilling hubs and authorities
- HITS for Clustering

Hyperlink-Induced Topic Search (HITS)

- In response to a query, instead of an ordered list of pages each meeting the query, find two sets of inter-related pages:
    - *Hub pages* are good lists of links on a subject.
        - e.g., "Bob's list of cancer-related links."
    - *Authority pages* occur recurrently on good hubs for the subject.
- Best suited for "broad topic" queries rather than for page-finding queries. (INFORMATIONAL QUERIES)
- Gets at a broader slice of common *opinion*.

Hubs and Authorities

- Thus, a good hub page for a topic *points* to many authoritative pages for that topic.
- A good authority page for a topic is *pointed* to by many good hubs for that topic.
- Circular definition - will turn this into an iterative computation.

Authorities

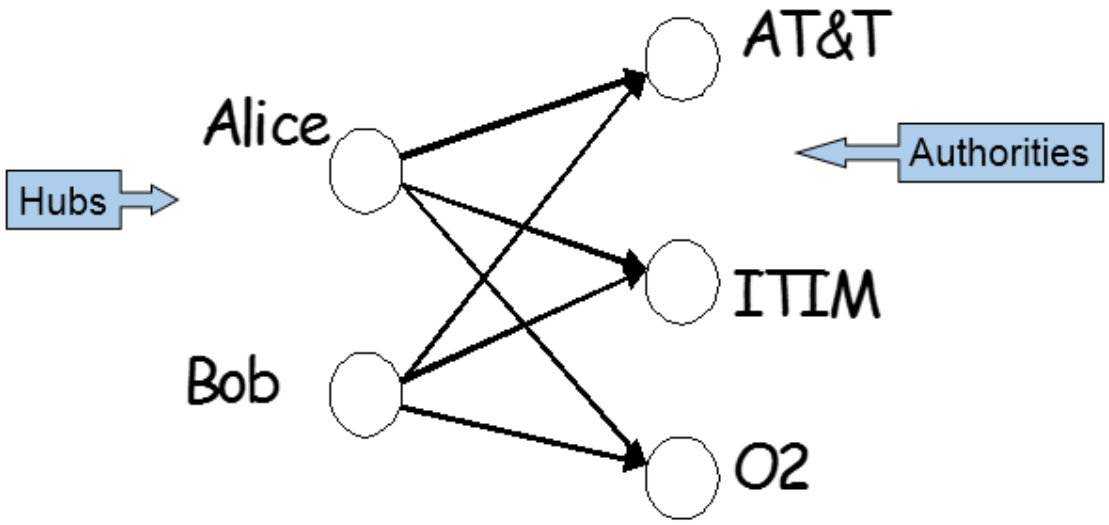
- *Authorities* are pages that are recognized as providing significant, trustworthy, and useful information on a topic.
- *In-degree* (number of pointers to a page) is one simple measure of authority.
- However in-degree treats all links as equal.

- Should links from pages that are themselves authoritative count more?

Hubs

- *Hubs* are index pages that provide lots of useful links to relevant content pages (topic authorities).
- Hub pages for IR are included in the course home page:
  - http://www.cs.utexas.edu/users/mooney/ir-course

The hope



Mobile telecom companies

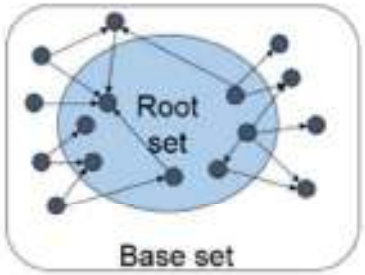
High-level scheme

- Extract from the web a base set of pages that *could* be good hubs or authorities.
- From these, identify a small set of top hub and authority pages;
  → iterative algorithm.

Base set

- Given text query (say ***browser***), use a text index to get all pages containing ***browser.***
  - Call this the root set of pages.
- Add in any page that either
  - points to a page in the root set, or
  - is pointed to by a page in the root set.
- Call this the base set.

Visualization



Get in-links (and out-links) from a *connectivity server*

Distilling hubs and authorities

- Compute, for each page *x* in the base set, a hub score *h(x)* and an authority score *a(x).*
- Initialize: for all *x, h(x)←1; a(x) ←1;*
- Iteratively update all *h(x), a(x);* Key
- After iterations
  - output pages with highest *h()* scores as top hubs
  - highest *a()* scores as top authorities.

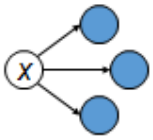
Base Limitations

- To limit computational expense:
  - Limit number of root pages to the top 200 pages retrieved for the query.
  - Limit number of "back-pointer" pages to a random set of at most 50 pages returned by a "reverse link" query.
- To eliminate purely navigational links:
  - Eliminate links between two pages on the same host.
- To eliminate "non-authority-conveying" links:
  - Allow only *m* ($m \cong 4$–8) pages from a given host as pointers to any individual page.

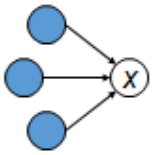
Iterative update

- Repeat the following updates, for all x:

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$



$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$



Scaling

- To prevent the *h()* and *a()* values from getting too big, can scale down after each iteration.
- Scaling factor doesn't really matter:
    - we only care about the *relative* values of the scores.

How many iterations?

- Claim: relative values of scores will converge after a few iterations:
    - in fact, suitably scaled, *h()* and *a()* scores settle into a steady state!
    - proof of this comes later.
- In practice, ~5 iterations get you close to stability.

Results

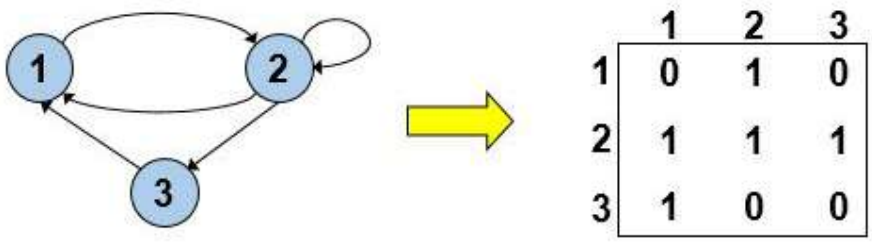
- Authorities for query: "Java"
    - java.sun.com
    - comp.lang.java FAQ
- Authorities for query "search engine"
    - Yahoo.com
    - Excite.com
    - Lycos.com
    - Altavista.com
- Authorities for query "Gates"
    - Microsoft.com
    - roadahead.com

Japan Elementary Schools

Hubs	Authorities
schools	The American School in Japan
LINK Page-13	The Link Page
"0–(.)ŠwŽ	‰ª©— § ¯ã"c¤ŠwŽƒzƒ‚ƒ‚ƒW
ª‰‚¤ŠwŽƒzƒ‚ƒ‚ƒW	Kids' Space
100 Schools Home Pages (English)	¯Ä©— § ¯Ä©‰•¯¤ŠwŽ
K-12 from Japan 10/...met and Education)	‹{©‹"ç ¯åŠw•¤®¤ŠwŽ
http://www...iglobe.ne.jp/~IKESAN	KEIMEI GAKUEN Home Page (Japanese)
‚‚‚¤ŠwŽ‚U¯N‚P ¯g•¯Œé	Shiranuma Home Page
©Š—¯ ¬— § ©Š—"Œ¤ŠwŽ	fuzoku-es.fukui-u.ac.jp
Koulutus ja oppilaitokset	welcome to Miasa E&J school
TOYODA HOMEPAGE	‡"Þ‰Œ § ‰¡•‰— § ´ †‰¤ŠwŽ‚ƒy
Education	http://www...p/~m_maru/index.html
Cay's Homepage(Japanese)	fukui haruyama-es HomePage
–y"!¤ŠwŽ‚ƒzƒ‚ƒ‚ƒW	Torisu primary school
UNIVERSITY	goo
‰J—ª¤ŠwŽ DRAGON97-TOP	Yakumo Elementary,Hokkaido,Japan
Á‰ª¤ŠwŽ‚T¯N‚P ¯g‚ƒzƒ‚ƒ‚ƒW	FUZOKU Home Page
¶µ" è¼ÁÁ© ¥á¥È¥å¡¼ ¥á¥È¥å¡¼	Kamishibun Elementary School...

Proof of convergence

- *n*×*n* <u>adjacency matrix</u> **A**:
  - each of the *n* pages in the base set has a row and column in the matrix.
  - Entry $A_{ij} = 1$ if page *i* links to page *j*, else = 0.



Hub/authority vectors

- View the hub scores *h()* and the authority scores *a()* as vectors with *n* components.
- Recall the iterative updates

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$

$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$

Rewrite in matrix form

- **h=Aa.**
- **a=A^{t}h.**

Recall A^t is the transpose of A.

Substituting, **h=AA^{t}h** and **a=A^{t}Aa**.

Thus, **h** is an eigenvector of **AA^t** and **a** is an eigenvector of **A^{t}A**.

Further, our algorithm is a particular, known algorithm for computing eigenvectors: the *power iteration* method.

Guaranteed to converge.

Finding Similar Pages Using Link Structure

- Given a page, *P*, let *R* (the root set) be *t* (e.g. 200) pages that point to *P*.
- Grow a base set *S* from *R*.
- Run HITS on *S*.
- Return the best authorities in *S* as the best similar-pages for *P*.
- Finds authorities in the "link neighbor-hood" of *P*.

Similar Page Results

- Given "honda.com"
    - toyota.com
    - ford.com
    - bmwusa.com
    - saturncars.com
    - nissanmotors.com
    - audi.com

- volvocars.com

HITS for Clustering

- An ambiguous query can result in the principal eigenvector only covering one of the possible meanings.
- Non-principal eigenvectors may contain hubs & authorities for other meanings.
- Example: "jaguar":
    - Atari video game (principal eigenvector)
    - NFL Football team (2nd non-princ. eigenvector)
    - Automobile (3rd non-princ. eigenvector)

Issues

- Topic Drift
    - Off-topic pages can cause off-topic "authorities" to be returned
        - E.g., the neighborhood graph can be about a "super topic"
- Mutually Reinforcing Affiliates
    - Affiliated pages/sites can boost each others' scores
        - Linkage between affiliated pages is not a useful signal

Resources

- IIR Chap 21
- http://www2004.org/proceedings/docs/1p309.pdf
- http://www2004.org/proceedings/docs/1p595.pdf
- http://www2003.org/cdrom/papers/refereed/p270/kamvar-270-xhtml/index.html
- http://www2003.org/cdrom/papers/refereed/p641/xhtml/p641-mccurley.html
- The WebGraph framework I: Compression techniques (Boldi et al. 2004)

Lecture # 39

- Search Computing

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Multi-domain queries with ranking
- Why Search Engines can't do it?
- Observed trends
- Search Computing
- The Search Computing "Manifesto"
- Search Computing architecture

Motivation: multi-domain queries with ranking

A class of queries search engines are not good at

- "Where can I attend an interesting scientific conference in my field and at the same time relax on a beautiful beach nearby?"
- "Retrieve jobs as Java developer in the Silicon Valley, nearby affordable fully-furnished flats, and close to good schools
- "Find a theater close to Union Square, San Francisco, showing a recent thriller movie, close to a steak house?"

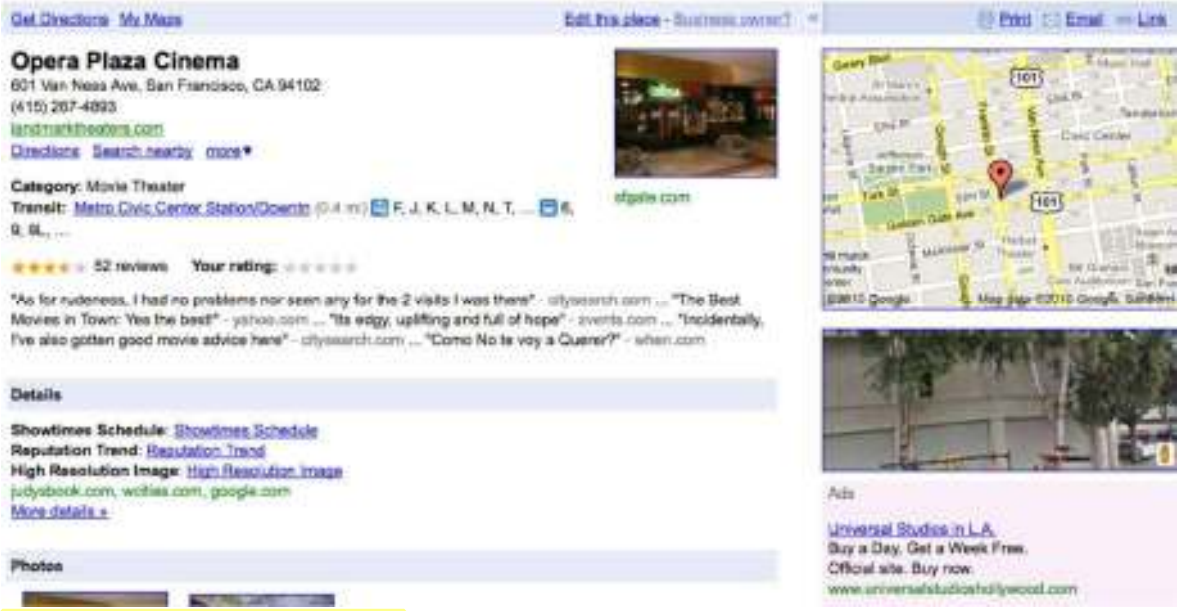
- With a complex notion of "best"
  - with many factors contributing to optimality
- Involving several different data sources
  - possibly hidden in the deep Web
  - typically returning ranked results (search services)
- With possibly articulated "join" conditions

Due to query complexity, not data heterogeneity or unavailability

Search For a Solution Using All Keywords

Split the task, and search for theaters first

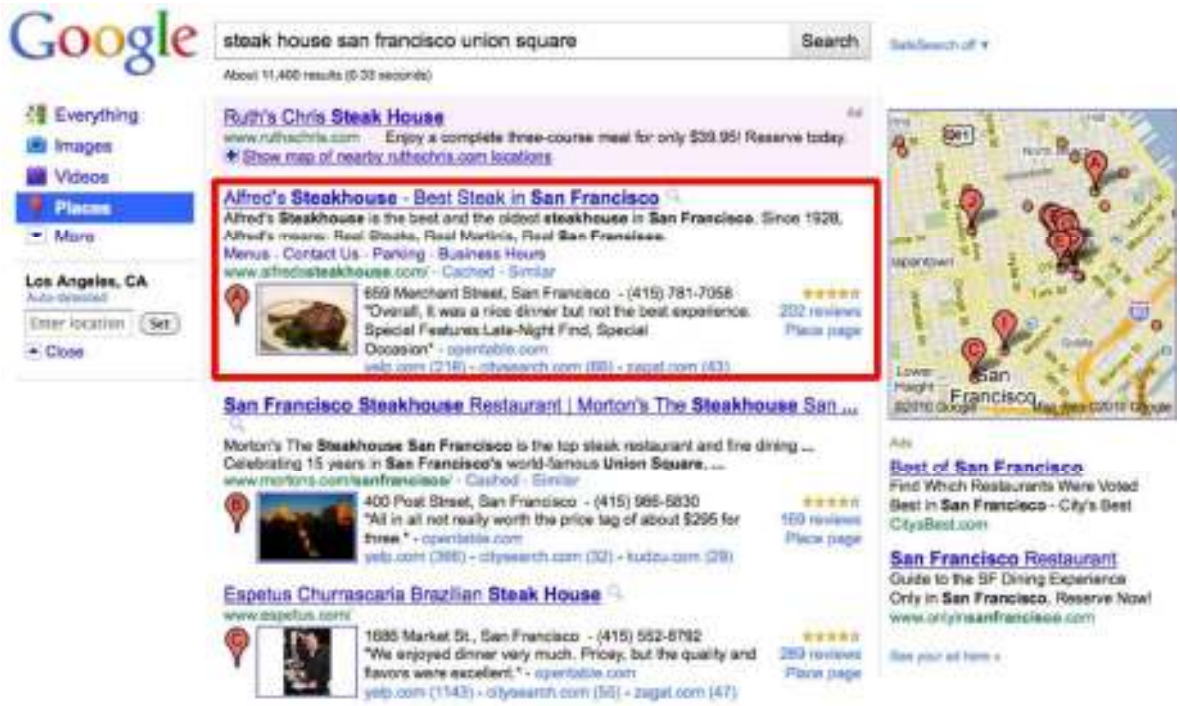
Inspect Theatre Details: Looks good…



But there's no thriller!

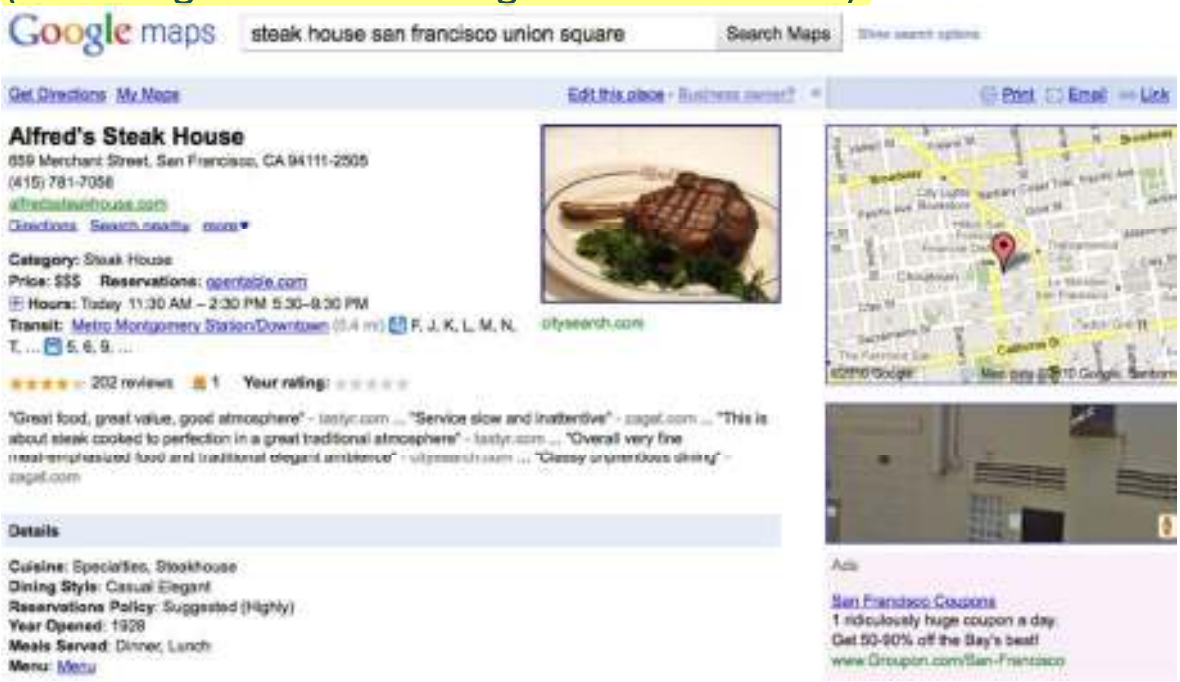
- Try another theater: Found! (The Next Three Days) close enough to Union square....

Independent search for steak house



Done! Close enough!

(data integration and ranking in the user's brain)



Motivating Examples – Why Search Engines can't do it?

- Query is about distinct domains that should be linked
- Query deals with multiple rankings, although hard to compute ("close" theatre, "recent" thriller, "good" steak house)

Note that

- enough data is on the Web but
- not on a single web page.

Observed trends

- More and more data sources become accessible through Web APIs (as services)
  - Sufrace & deep Web
- Data sources are often coupled with **search** APIs
- Publishing of structured and interconnected data is becoming popular (*Linked Open Data*)

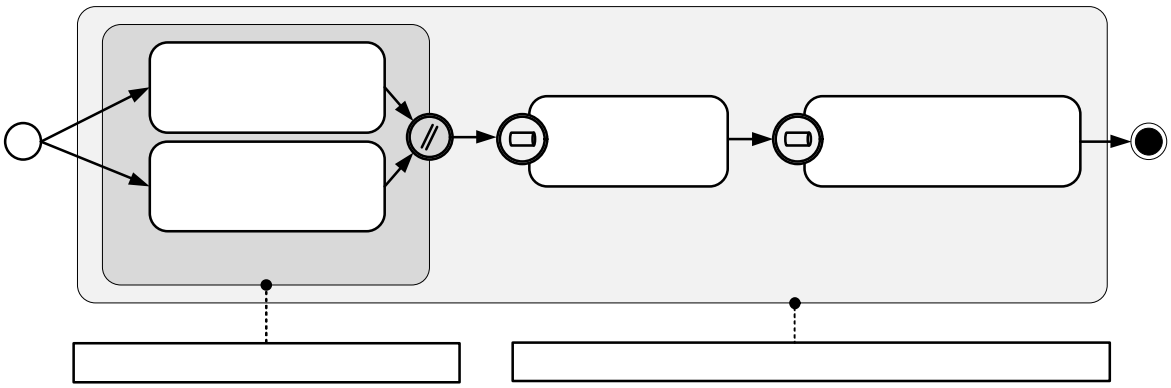
Opportunity for building **focused search systems composing results of several data source**

- easy-to-build, easy-to-query, easy-to-maintain, easy-to-scale...
- covering the functionalities of vertical search systems (e.g. "expedia", "amazon") on **more focused application domains**
  - (e.g. **localized** real estate or leasure planning, **sector-specific** job market offers, support of biomedical **research**, ...)

Search Computing = service composition "on demand"

- Composition **abstractions** should emphasize few aspects:
  - service invocations
  - fundamental operations (parallel invocations, joins, pipelining, …)
  - global constraints on execution
- Data composition should be search-driven
  - aimed at producing **few top results** very fast

A house in a walk able area, close to public transportation and located in a pleasant neighborhood



The Search Computing "Manifesto"

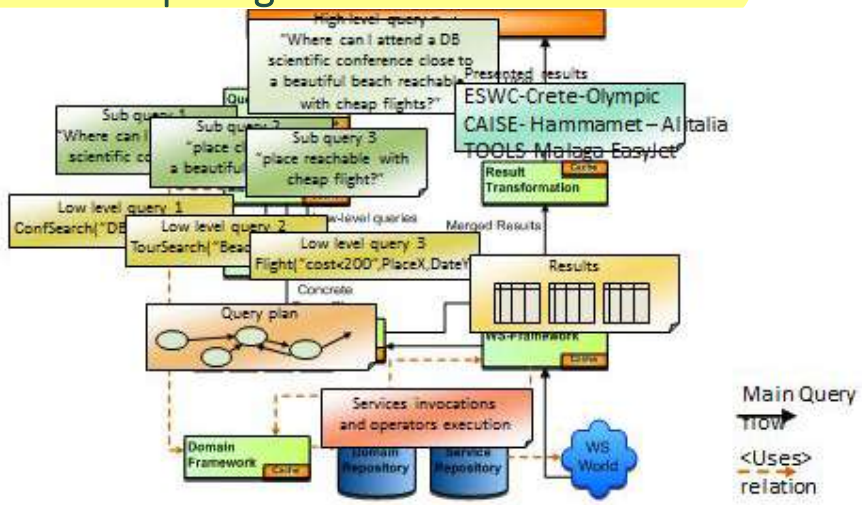
- Build theories, methods, and tools to support search-oriented multi-domain queries

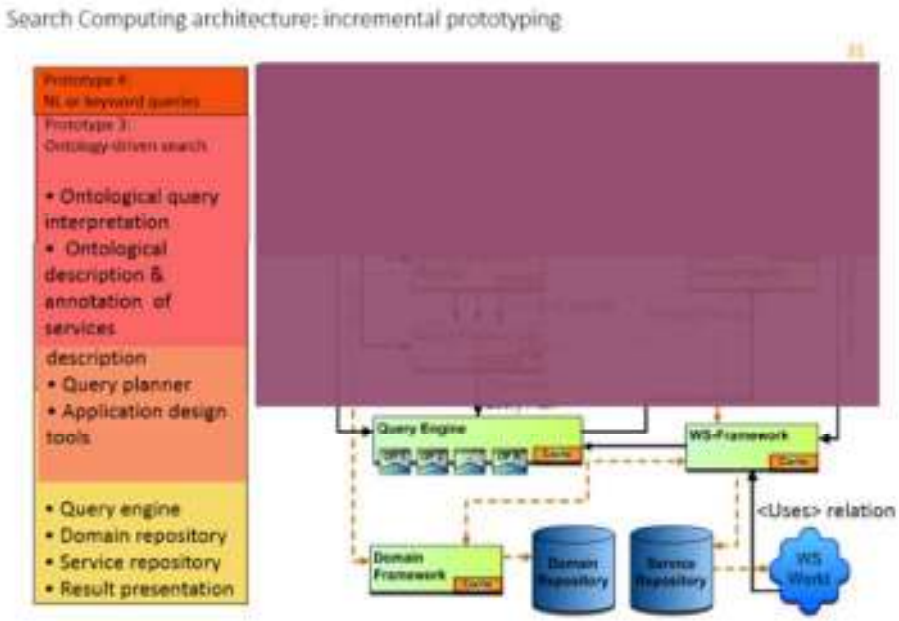
Given a **multi-domain query** over a set of **search services**

- Build global answers by **combining** data from each service
- Rank global answers according to a **global ranking** and output results in ranking order
- Support user-friendly query **formulation** and browsing of results
- Include new domains while the **search process** proceeds
- Possibly change the relative weight of each partial ranking

"*Searching via interactive/dynamic mashups of ranked data sources*"

Search Computing architecture: overall view




Search Computing architecture: incremental prototyping
• Ontological query interpretation
• Ontological description & annotation of services
description
• Query planner
• Application design tools
• Query engine
• Domain repository
• Service repository
• Result presentation
Query Engine
WS-Framework
<Uses> relation
Domain Framework
Domain Repository
Service Repository
WS World

Lecture # 40

- Top-k Query Processing

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

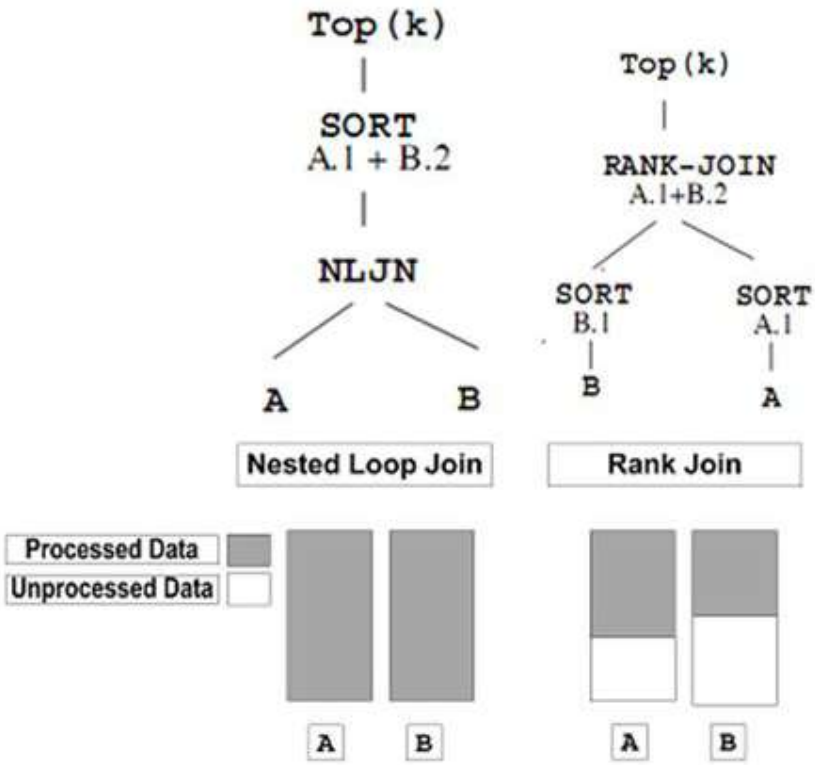
- Top-k Query Processing
- Simple Database model
- Fagin's Algorithm
- Threshold Algorithm
- Comparison of Fagin's and Threshold Algorithm

Top-k Query Processing

Optimal aggregation algorithms for middleware

Ronald Fagin, Amnon Lotem, and Moni Naor

Top-k vs Nested Loop Query



Rank join vs nested loop join.

Example

- Simple database model
  - Simple query
- Explaining Fagin's Algorithm (FA)
  - Finding top-k with FA
- Explaining Threshold Algortihm (TA)
  - Finding top-k with TA

Example – Simple Database model

M

Object ID	Attribute 1	Attribute 2
a	0.9	0.85
b	0.8	0.7
c	0.72	0.2
d	0.6	0.9
.	.	.
.	.	.
.	.	.
.	.	.

N

Sorted L_1	Sorted L_2
(a, 0.9)	(d, 0.9)
(b, 0.8)	(a, 0.85)
(c, 0.72)	(b, 0.7)
.	.
.	.
.	.
.	.
(d, 0.6)	(c, 0.2)

Example – Fagin's Algorithm

STEP 1

- Read attributes from every sorted list
- Stop when *k* objects have been seen in common from all lists

L_1	L_2
(a, 0.9)	(d, 0.9)
(b, 0.8)	(a, 0.85)
(c, 0.72)	(b, 0.7)
.	.
.	.
.	.
.	.
(d, 0.6)	(c, 0.2)

ID	A_1	A_2	$Min(A_1,A_2)$
a	0.9	0.85	
d		0.9	
b	0.8	0.7	
c	0.72		

Example – Fagin's Algorithm

STEP 2

- Random access to find missing grades

L_1	L_2
(a, 0.9)	(d, 0.9)
(b, 0.8)	(a, 0.85)
(c, 0.72)	(b, 0.7)
.	.
.	.
.	.
.	.
(d, 0.6)	(c, 0.2)

ID	A_1	A_2	$Min(A_1,A_2)$
a	0.9	0.85	
d	0.6	0.9	
b	0.8	0.7	
c	0.72	0.2	

Example – Fagin's Algorithm

STEP 3

- Compute the grades of the seen objects.
- Return the k highest graded objects.

L_1	L_2
(a, 0.9)	(d, 0.9)
(b, 0.8)	(a, 0.85)
(c, 0.72)	(b, 0.7)
.	.
.	.
.	.
.	.
(d, 0.6)	(c, 0.2)

ID	A_1	A_2	$Min(A_1,A_2)$
a	0.9	0.85	0.85
d	0.6	0.9	0.6
b	0.8	0.7	0.7
c	0.72	0.2	0.2

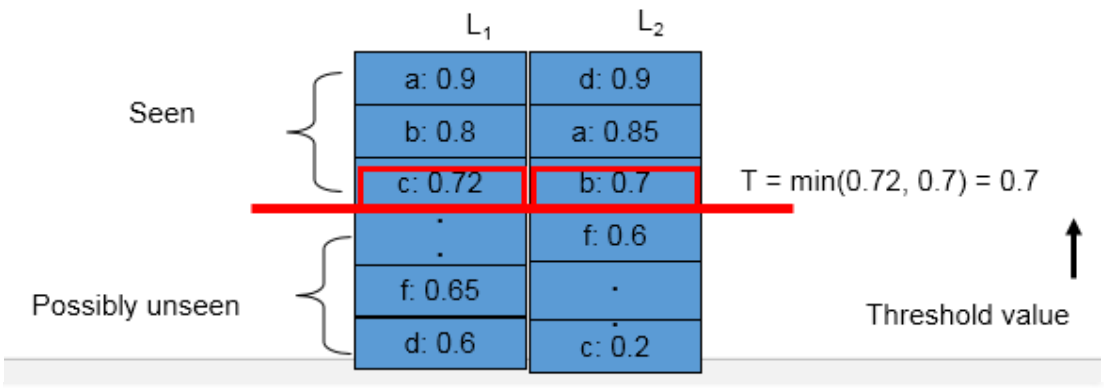
New Idea !!! Threshold Algorithm (TA)

Read all grades of an object once seen from a sorted access

- No need to wait until the lists give *k* common objects

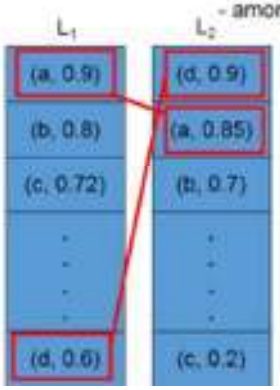
Do sorted access (and corresponding random accesses) until you have seen the top k answers.

- How do we know that grades of seen objects are higher than the grades of unseen objects ?
- Predict maximum possible grade unseen objects:



Example – Threshold Algorithm

Step 1: - parallel sorted access to each list
For each object seen:
- get all grades by random access
- determine Min(A1,A2)
- amongst 2 highest seen ? keep in buffer



ID	A_1	A_2	$Min(A_1,A_2)$
a	0.9	0.85	0.85
d	0.6	0.9	0.6

Example – Threshold Algorithm

Step 2: - Determine threshold value based on objects currently seen under sorted access. T = min(L1, L2)
- 2 objects with overall grade ≥ threshold value ? stop else go to next entry position in sorted list and repeat step 1

L_1	L_2
a: 0.9	d: 0.9
b: 0.8	a: 0.85
c: 0.72	b: 0.7
.	.
.	.
.	.
.	.
d: 0.6	c: 0.2

ID	A_1	A_2	$Min(A_1,A_2)$
a	0.9	0.85	0.85
d	0.6	0.9	0.6

T = min(0.9, 0.9) = 0.9

Example – Threshold Algorithm

Step 1 (Again): - parallel sorted access to each list
For each object seen:
- get all grades by random access
- determine Min(A1,A2)
- amongst 2 highest seen ? keep in buffer

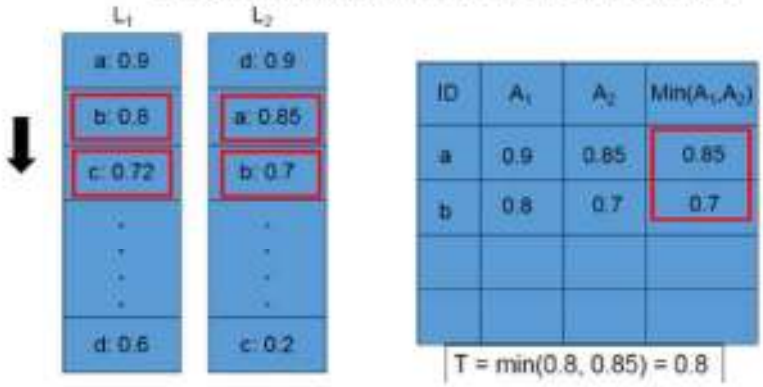
L_1	L_2
(a, 0.9)	(d, 0.9)
(b, 0.8)	(a, 0.85)
(c, 0.72)	(b, 0.7)
.	.
.	.
.	.
.	.
(d, 0.6)	(c, 0.2)

ID	A_1	A_2	$Min(A_1,A_2)$
a	0.9	0.85	0.85
d	0.6	0.9	0.6
b	0.8	0.7	0.7

Example – Threshold Algorithm

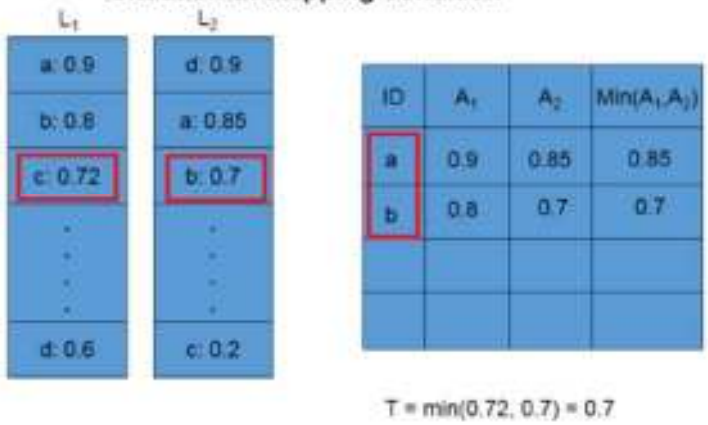
Step 2 (Again): - Determine threshold value based on objects currently seen. T = min(L1, L2)

- 2 objects with overall grade ≥ threshold value ? stop else go to next entry position in sorted list and repeat step 1



Example – Threshold Algorithm

Situation at stopping condition



Comparison of Fagin's and Threshold Algorithm

- TA sees less objects than FA
  - TA stops at least as early as FA
    - When we have seen *k* objects in common in FA, their grades are higher or equal than the threshold in TA.
- TA may perform more random accesses than FA

- In TA, (m-1) random accesses for each object
- In FA, Random accesses are done at the end, only for missing grades

- TA requires only bounded buffer space (k)
  - At the expense of more random seeks
  - FA makes use of unbounded buffers

The best algorithm

Which algorithm is the best: TA, FA??

- Define "best"
  - middleware cost
  - concept of instance optimality
- Consider:
  - wild guesses
  - aggregation functions characteristics
    - Monotone, strictly monotone, strict
  - database restrictions
    - distinctness property

The best algorithm: aggregation functions

- Aggregation function t combines object grades into object's overall grade:

  $x_1,\ldots,x_m$ $\quad$ $t(x_1,\ldots,x_m)$
- Monotone :

  $t(x_1,\ldots,x_m) \le t(x'_1,\ldots,x'_m)$ if $x_i \le x'_i$ for every i
- Strictly monotone:

  $t(x_1,\ldots,x_m) < t(x'_1,\ldots,x'_m)$ if $x_i < x'_i$ for every i
- Strict:

  $t(x_1,\ldots,x_m) = 1$ precisely when $x_i = 1$ for every i

Extending TA

- What if sorted access is restricted ? e.g. use distance database
  - TA_z
  - What if random access not possible? e.g. web search engine
    - No Random Access Algorithm
  - What if we want only the approximate top k objects?
    - TA_θ
- What if we consider relative costs of random and sorted access?
  - Combined Algorithm (between TA and NRA)

Taxonomy of Top-k Joins

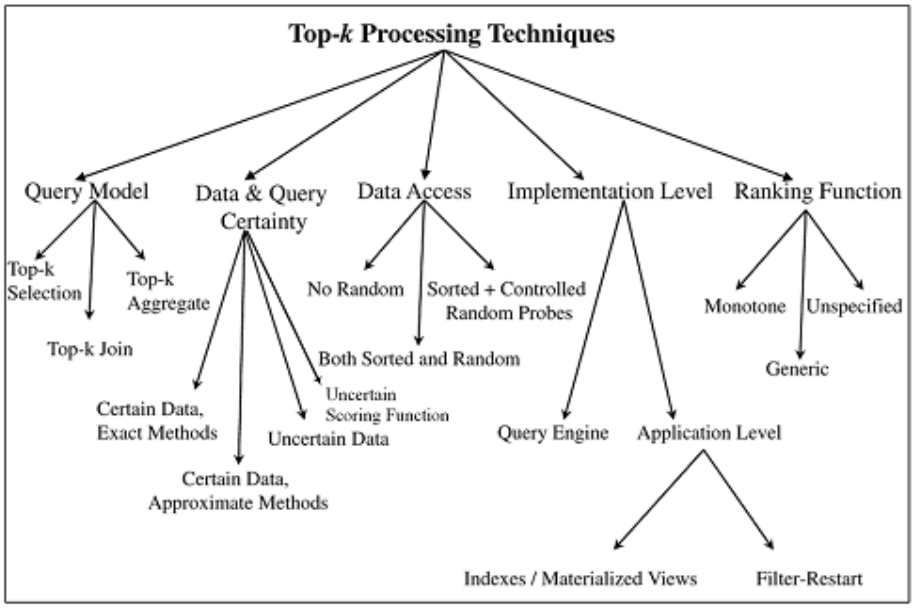


Figure 2.2: Taxonomy of Top-K Joins

Lecture # 41

- Clustering

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

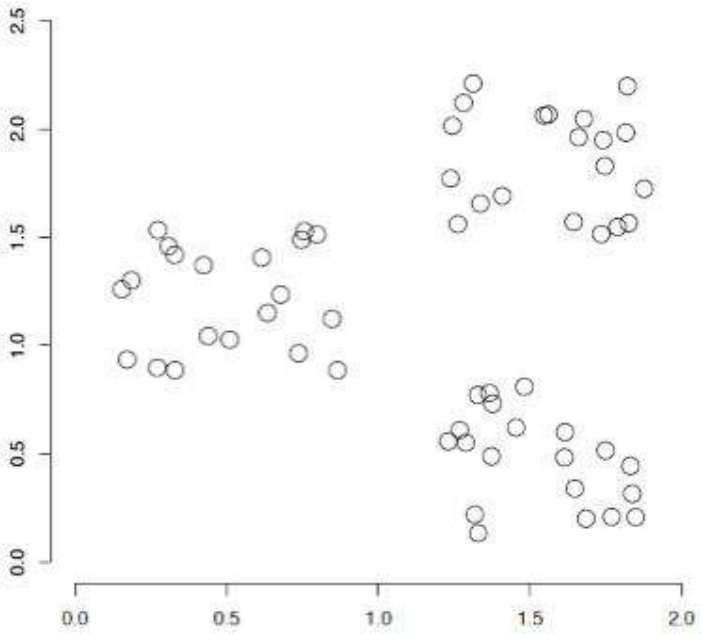
Outline

- What is clustering?
- Improving search recall
- Issues for clustering
- Notion of similarity/distance
- Hard vs. soft clustering
- Clustering Algorithms

What is clustering?

- Clustering: the process of grouping a set of objects into classes of similar objects
    - Documents within a cluster should be similar.
    - Documents from different clusters should be dissimilar.
- The commonest form of *unsupervised learning*
    - Unsupervised learning = learning from raw data, as opposed to supervised data where a classification of examples is given
    - A common and important task that finds many applications in IR and other places

A data set with clear cluster structure

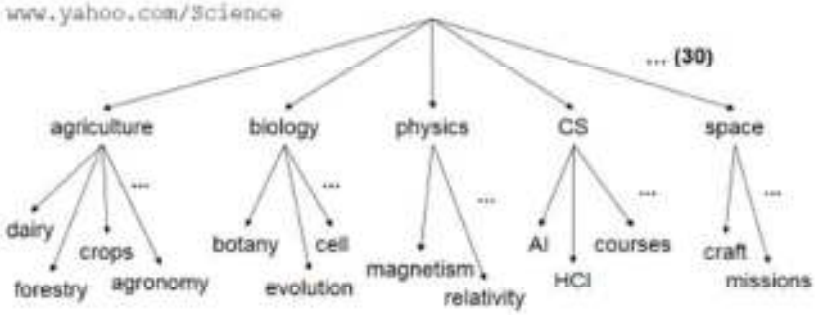


- How would you design an algorithm for finding the three clusters in this case?

Applications of clustering in IR

- **Whole corpus analysis/navigation**
    - **Better user interface: search without typing**
- For improving recall in search applications
    - Better search results (like pseudo RF)
- For better navigation of search results
    - Effective "user recall" will be higher
- For speeding up vector space retrieval
    - Cluster-based retrieval gives faster search

Yahoo! Hierarchy *isn't* clustering but *is* the kind of output you want from clustering



Google News: automatic clustering gives an effective news presentation metaphor

Applications of clustering in IR

- Whole corpus analysis/navigation
    - Better user interface: search without typing
- **For improving recall in search applications**
    - **Better search results (like pseudo RF)**
- For better navigation of search results
    - Effective "user recall" will be higher
- For speeding up vector space retrieval
    - Cluster-based retrieval gives faster search

For improving search recall

- *Cluster hypothesis* - Documents in the same cluster behave similarly with respect to relevance to information needs
- Therefore, to improve search recall:
    - Cluster docs in corpus a priori
    - When a query matches a doc *D*, also return other docs in the cluster containing *D*
- Hope if we do this: The query "car" will also return docs containing *automobile*
    - Because clustering grouped together docs containing *car* with those containing *automobile.*

Applications of clustering in IR

- Whole corpus analysis/navigation
    - Better user interface: search without typing
- For improving recall in search applications

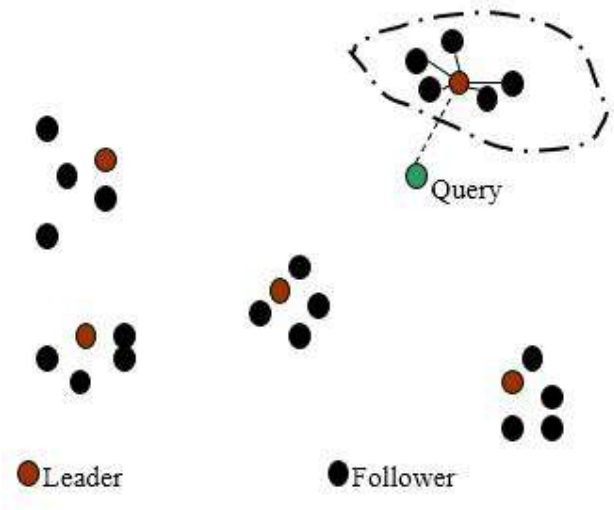
- Better search results (like pseudo RF)
- **For better navigation of search results**
    - **Effective "user recall" will be higher**
- For speeding up vector space retrieval
    - Cluster-based retrieval gives faster search

yippy.com – grouping search results

Applications of clustering in IR

- Whole corpus analysis/navigation
    - Better user interface: search without typing
- For improving recall in search applications
    - Better search results (like pseudo RF)
- For better navigation of search results
    - Effective "user recall" will be higher
- **For speeding up vector space retrieval**
    - **Cluster-based retrieval gives faster search**

Visualization



Issues for clustering

- Representation for clustering
    - Document representation
        - Vector space? Normalization?
    - Need a notion of similarity/distance
- How many clusters?
    - Fixed a priori?
    - Completely data driven?
        - Avoid "trivial" clusters - too large or small
            - If a cluster's too large, then for navigation purposes you've wasted an extra user click without whittling down the set of documents much.

Notion of similarity/distance

- Ideal: semantic similarity.
- Practical: term-statistical similarity (docs as vectors)
    - Cosine similarity
    - For many algorithms, easier to think in terms of a *distance* (rather than similarity) between docs.
    - We will mostly speak of Euclidean distance
        - But real implementations use cosine similarity

Hard vs. soft clustering

- Hard clustering: Each document belongs to exactly one cluster
    - More common and easier to do
- Soft clustering: A document can belong to more than one cluster.

- Makes more sense for applications like creating brows able hierarchies
- You may want to put a pair of sneakers in two clusters: (i) sports apparel and (ii) shoes
- You can only do that with a soft clustering approach.

Clustering Algorithms

- Flat algorithms
  - Usually start with a random (partial) partitioning
  - Refine it iteratively
    - *K* means clustering
    - (Model based clustering)
- Hierarchical algorithms
  - Bottom-up, agglomerative
  - (Top-down, divisive)

Partitioning Algorithms

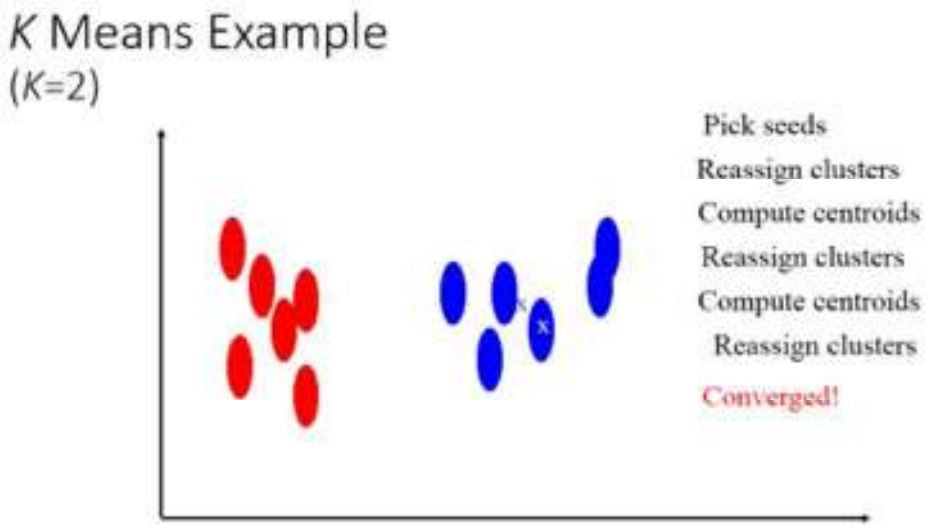
- Partitioning method: Construct a partition of *n* documents into a set of *K* clusters
- Given: a set of documents and the number *K*
- Find: a partition of *K* clusters that optimizes the chosen partitioning criterion
  - Globally optimal
    - Intractable for many objective functions
    - Ergo, exhaustively enumerate all partitions
  - Effective heuristic methods: *K*-means and *K*-medoids algorithms

K-Means

- Assumes documents are real-valued vectors.
- Clusters based on *centroids* (aka the *center of gravity* or mean) of points in a cluster, *c*:

$$\vec{\mu}(c) = \frac{1}{|c|}\sum_{\vec{x}\in c}\vec{x}$$

- Reassignment of instances to clusters is based on distance to the current cluster centroids.
  - (Or one can equivalently phrase it in terms of similarities)



Termination conditions

- Several possibilities, e.g.,
    - A fixed number of iterations.
    - Doc partition unchanged.
    - Centroid positions don't change.

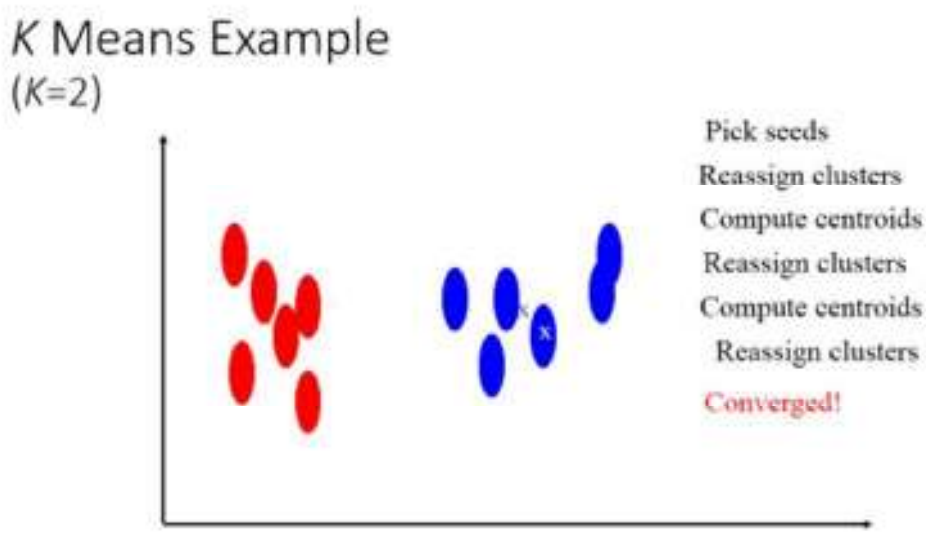
Does this mean that the docs in a cluster are unchanged?

Convergence

- Why should the *K*-means algorithm ever reach a *fixed point*?
    - A state in which clusters don't change.
- *K*-means is a special case of a general procedure known as the *Expectation Maximization (EM) algorithm*.
    - EM is known to converge.
    - Number of iterations could be large.
        - But in practice usually isn't

Convergence of *K*-Means

- Residual Sum of Squares (RSS), a goodness measure of a cluster, is the sum of squared distances from the cluster centroid:
  - $RSS_j = \Sigma_i |d_i - c_j|^2$ (sum over all d_i in cluster *j*)
- $RSS = \Sigma_j RSS_j$
- Reassignment monotonically decreases RSS since each vector is assigned to the closest centroid.
- Recomputation also monotonically decreases each RSS_j because …



Convergence of *K*-Means

- Residual Sum of Squares (RSS), a goodness measure of a cluster, is the sum of squared distances from the cluster centroid:
  - $RSS_j = \Sigma_i |d_i - c_j|^2$ (sum over all d_i in cluster *j*)
- $RSS = \Sigma_j RSS_j$
- Reassignment monotonically decreases RSS since each vector is assigned to the closest centroid.
- Recomputation also monotonically decreases each RSS_j because …

Seed Choice

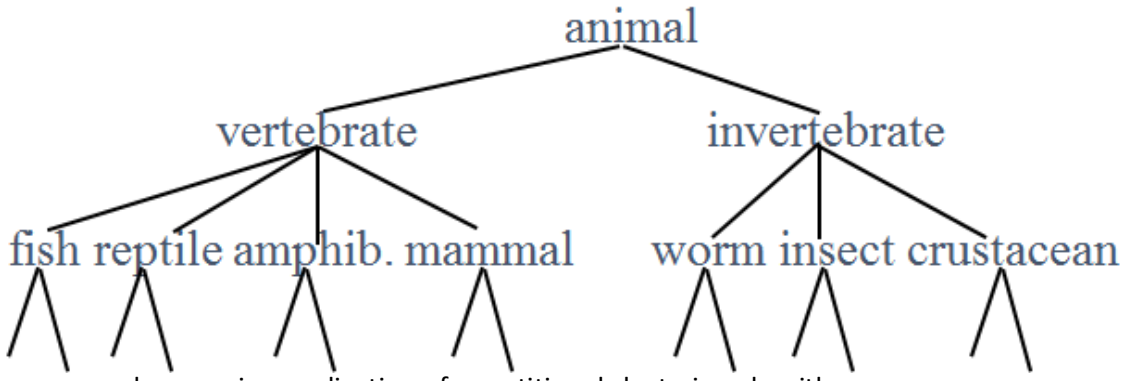
- Results can vary based on random seed selection.
- Some seeds can result in poor convergence rate, or convergence to sub-optimal clusterings.
  - Select good seeds using a heuristic (e.g., doc least similar to any existing mean)
  - Try out multiple starting points
  - Initialize with the results of another method.

Example showing sensitivity to seeds

In the above, if you start with B and E as centroids you converge to {A,B,C} and {D,E,F}
If you start with D and F you converge to {A,B,D,E} {C,F}

Hierarchical Clustering

- Build a tree-based hierarchical taxonomy (*dendrogram*) from a set of documents.



- One approach: recursive application of a partitional clustering algorithm.

Final word and resources

- In clustering, clusters are inferred from the data without human input (unsupervised learning)
- However, in practice, it's a bit less clear: there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of documents, . . .

Resources

- IIR 16 except 16.5
- IIR 17.1–17.3

Lecture # 42

- Classification

ACKNOWLEDGEMENTS

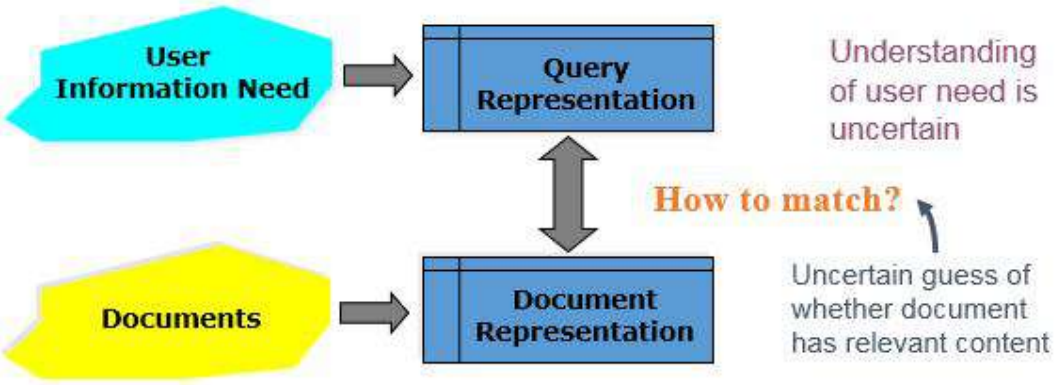
The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Why probabilities in IR?
- Document Classification
- Bayes' Rule For Text Classification
- Bernoulli Random Variables
- Smoothing Function

Why probabilities in IR?



In traditional IR systems, matching between each document and query is attempted in a semantically imprecise space of index terms. Probabilities provide a principled foundation for uncertain reasoning. *Can we use probabilities to quantify our uncertainties?*

The document ranking problem

- We have a collection of documents
- User issues a query
- A list of documents needs to be returned
- **Ranking method is the core of an IR system:**
    - **In what order do we present documents to the user?**
    - We want the "best" document to be first, second best second, etc….
- **Idea: Rank by probability of relevance of the document w.r.t. information need**
    - $P(R=1|\text{document}_i, \text{query})$

Classification Methods (1)

- Manual classification
    - Used by the original Yahoo! Directory
    - Looksmart, about.com, ODP, PubMed
    - Accurate when job is done by experts
    - Consistent when the problem size and team is small
    - Difficult and expensive to scale
        - Means we need automatic classification methods for big problems

Classification Methods (2)

- Hand-coded rule-based classifiers
    - Rules are defined by subject experts.
    - Commercial systems have complex query languages
    - Accuracy is can be high if a rule has been carefully refined over time by a subject expert
    - Building and maintaining these rules is expensive

Classification Methods (3): Supervised learning

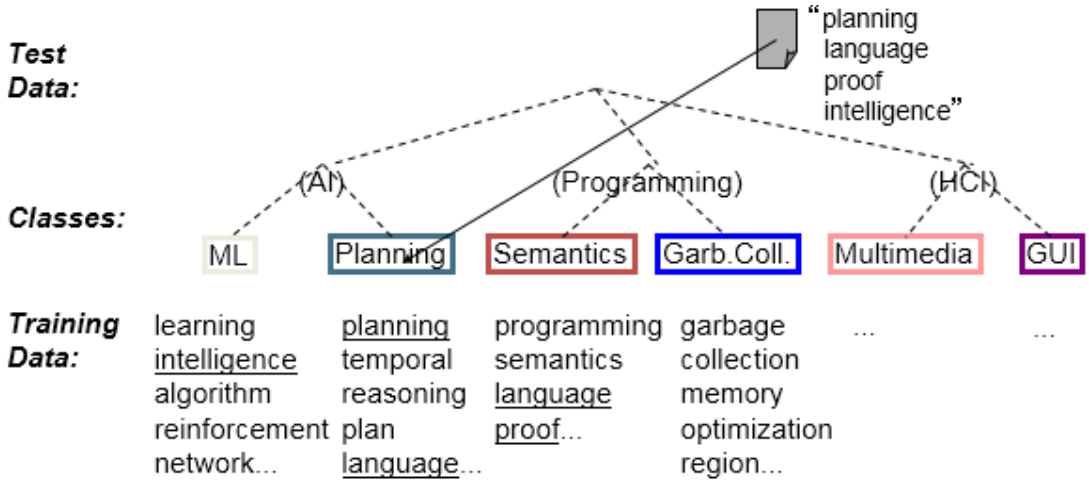
- Given:
    - A document *d*
    - A fixed set of classes:

$C = \{c_1, c_2, \ldots, c_J\}$

    - A <u>training set</u> *D* of documents each with a label in *C*
- Determine:
    - A learning method or algorithm which will enable us to learn a classifier *γ*
    - For a test document *d*, we assign it the class

$\gamma(d) \in C$

Document Classification



Bayes' Rule For Text Classification

- For a document ***D*** and a class ***C***

$$P(C,D) = P(C \mid D)P(D) = P(D \mid C)P(C)$$

$$\boxed{P(C \mid D) = \frac{P(D \mid C)P(C)}{P(D)}}$$

Bayes' Rule For Text Classification

- There are "V" terms in the vocabulary.

  Vocabulary = $|V|$
- The Document "D" contain "k" words.

K << $|V|$

D = <w_1, w_2, ..., wk>

$$P(C \mid D) = \frac{P(D \mid C)P(C)}{P(D)}$$

Bayes' Rule For Text Classification

$$D = <w_1, w_2, \ldots\ldots, wk>$$

$$<w_1, w_2, \ldots\ldots, wk> = P(w_1/C)\, P(w_2/C)\, P(w_3/C) \ldots\ldots\ldots P(w_k/C)$$

- $$P(C \mid <w_1, w_2, \ldots\ldots, wk>) = \frac{P(<w_1, w_2, \ldots\ldots, wk> \mid C)\, P(C)}{P(<w_1, w_2, \ldots\ldots, wk>)}$$

$$<C_1, C_2, \ldots\ldots, C_j>$$

Bernoulli Random Variables

- There are a_1, a_2, a_3 .....,a_L. "L" which are not present in the document "D" from the vocabulary $|V|$.

$<w_1, w_2, \ldots\ldots, wk> = \{P(\bar{a}_1 \mid C)\, P(\bar{a}_2 \mid C)\, P(\bar{a}_3 \mid C) \ldots\ldots\ldots P(\bar{a}_L \mid C)\}]$

$<w_1, w_2, \ldots\ldots, wk> = [1 - \{P(a_1 \mid C)\, P(a_2 \mid C)\, P(a_3 \mid C) \ldots\ldots\ldots P(a_L \mid C)\}]$

Training Set of Documents

Doc ID	Cheap	Buy	Banking	Dinner	The	Class
1	0	0	0	0	1	Not spam
2	1	0	1	0	1	Spam
3	0	0	0	0	1	Not spam
4	1	0	1	0	1	Spam
5	1	1	0	0	1	spam
6	0	0	1	0	1	Not spam
7	0	1	1	0	1	Not spam
8	0	0	0	0	1	Not spam
9	0	0	0	0	1	Not spam
10	1	1	0	1	1	Not spam

Probability of Training Set

$$P(spam \mid d) = \frac{P(d \mid spam)P(spam)}{P(d)}$$

$$P(Not-spam \mid d) = \frac{P(d \mid Not-spam)P(Not-spam)}{P(d)}$$

Training Set of Documents

Doc ID	Cheap	Buy	Banking	Dinner	The	Class
1	0	0	0	0	1	Not spam
2	1	0	1	0	1	Spam
3	0	0	0	0	1	Not spam
4	1	0	1	0	1	Spam
5	1	1	0	0	1	spam
6	0	0	1	0	1	Not spam
7	0	1	1	0	1	Not spam
8	0	0	0	0	1	Not spam
9	0	0	0	0	1	Not spam
10	1	1	0	1	1	Not spam

Probability of Training Set

- Input Documents <buy, cheap, dinner>

$P(d|spam) = P(buy|spam)P(cheap|spam)\ P(dinner|spam) \times P(\overline{banking}|spam)P(\overline{the}|\ spam)$

$P(\overline{banking}|spam)P(\overline{the}|\ spam) = \{1 - P(banking|spam)\}\ \{1 - P(the|\ spam)\}$

- 1-P(the|spam) = 0 so the whole expression would become zero.
- P(the|spam) = no. of docs in spam class having the/total no. of spam docs
- So smoothing is required

Smoothing Function

$$Smoothing = \frac{1 + numerator}{1 + denominator}$$

$$Smoothing = \frac{1 + numerator}{\#\ of\ classes}$$

Another version of Naïve Bayse

- In previous version we consider the document frequency of a term i.e. in how many documents is it present.
- In this newer version we consider term frequency in the document and in the corpus

Training Set of Documents

Doc ID	Cheap	Buy	Banking	Dinner	The	Class
1	0	0	0	0	2	Not spam
2	3	0	1	0	1	Spam
3	0	0	0	0	1	Not spam
4	2	0	3	0	2	Spam
5	5	2	0	0	1	spam
6	0	0	1	0	1	Not spam
7	0	1	1	0	1	Not spam
8	0	0	0	0	1	Not spam
9	0	0	0	0	1	Not spam
10	1	1	0	1	2	Not spam

Probability of Training Set

- Input Documents <buy, cheap, dinner>

$$P(spam \mid d) = \frac{P(d \mid spam)P(spam)}{P(d)}$$

$$P(Not-spam \mid d) = \frac{P(d \mid Not-spam)P(Not-spam)}{P(d)}$$

$$P(buy, cheap, dinner|spam) = P(buy|spam)P(cheap|spam)P(dinner|spam)$$

Probability of Training Set

Doc ID	Cheap	Buy	Banking	Dinner	the	Class
1	0	0	0	0	2	Not spam
2	3	0	1	0	1	Spam
3	0	0	0	0	1	Not spam
4	2	0	3	0	2	Spam
5	3	2	0	0	1	spam
6	0	0	1	0	1	Not spam
7	0	1	1	0	1	Not spam
8	0	0	0	0	1	Not spam
9	0	0	0	0	1	Not spam
10	1	1	0	1	2	Not spam

Input doc is "buy cheap dinner" P(buy|spam) = 2/(5+7+8) Unigram would consider only the terms in the document and Will not consider the absent terms therefore, in this case we are

i) Considering 'k' terms not 'v'.
ii) We are considering "tf" not Just the presence or absence.
iii) Less no. of multiplications.

This model is better than the one discussed earlier as it considers frequencies of terms instead of their presence.

Naïve Bayse Characteristics

- **Underflow** error may occur in both cases when we implement it on a machine.
- We take log of both sides which converts multiplication into addition and avoids underflow
- **Feature Selection**: cut down the size of vocabulary.
- Consider the subset of terms i.e. drop rare terms and stop words.

Lecture # 43

- Classification

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

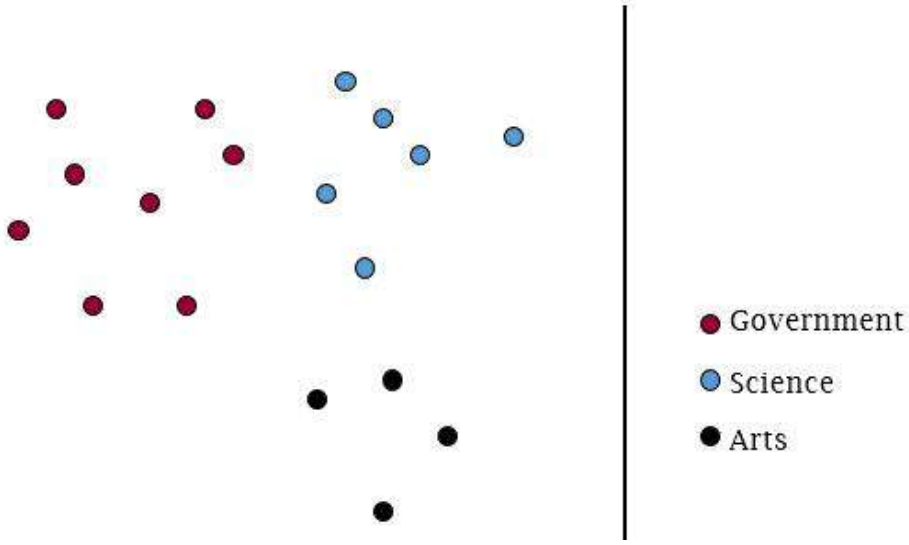
Outline

- Rochio classification
- K Nearest neighbors
- Nearest-Neighbor Learning
- kNN decision boundaries
- Bias vs. variance

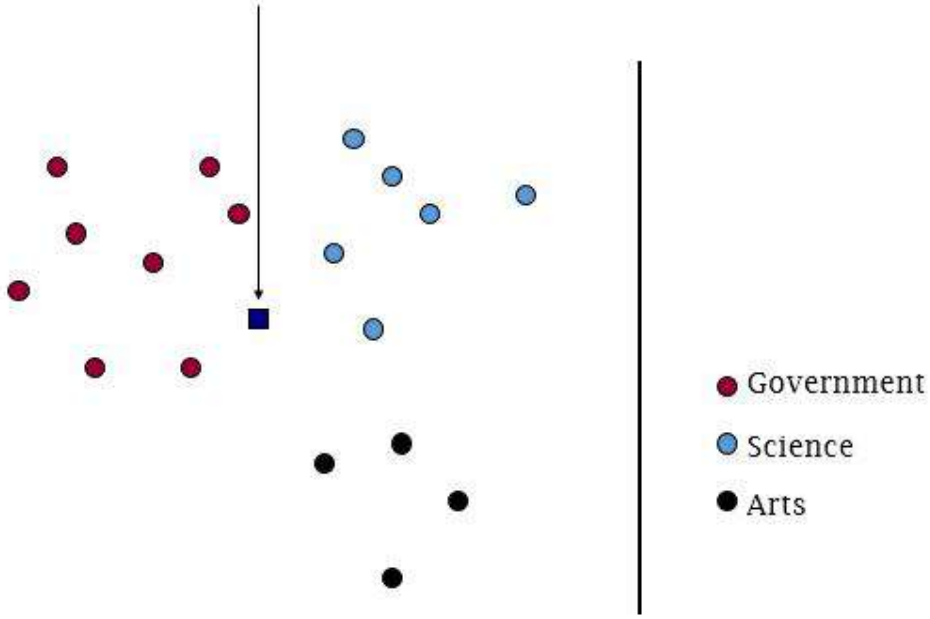
Classification Using Vector Spaces

- In vector space classification, training set corresponds to a labeled set of points (equivalently, vectors)
- Premise 1: Documents in the same class form a contiguous region of space
- Premise 2: Documents from different classes don't overlap (much)
- Learning a classifier: build surfaces to delineate classes in the space

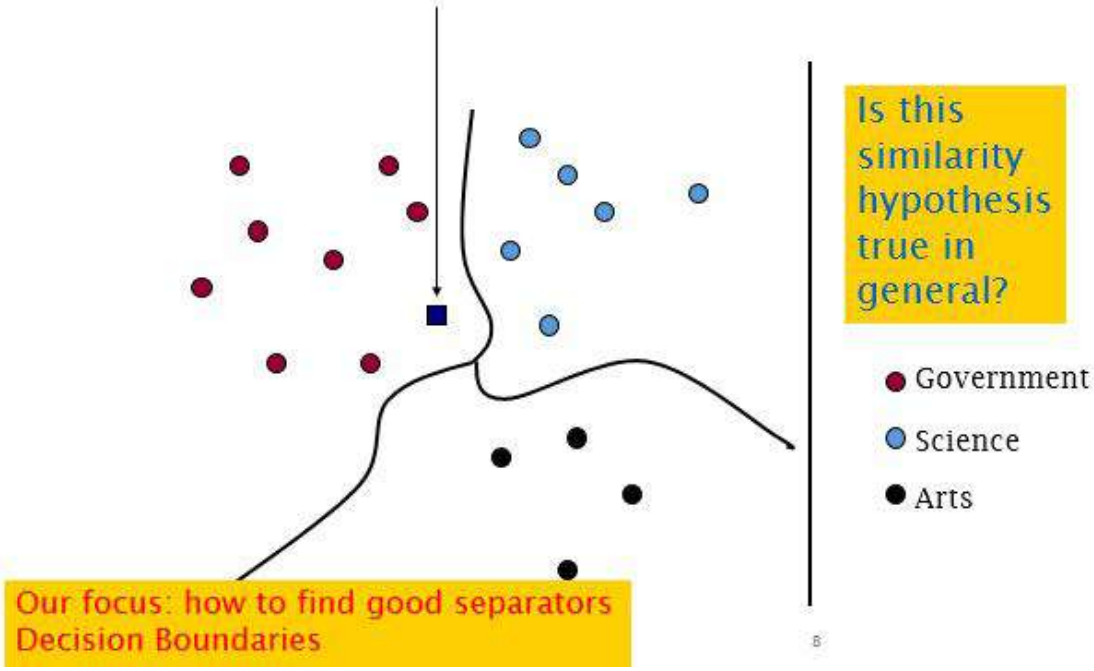
Documents in a Vector Space



Test Document of what class?

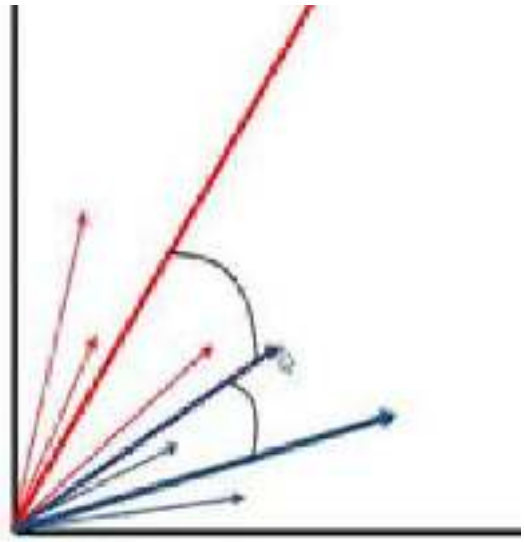


Test Document = Government



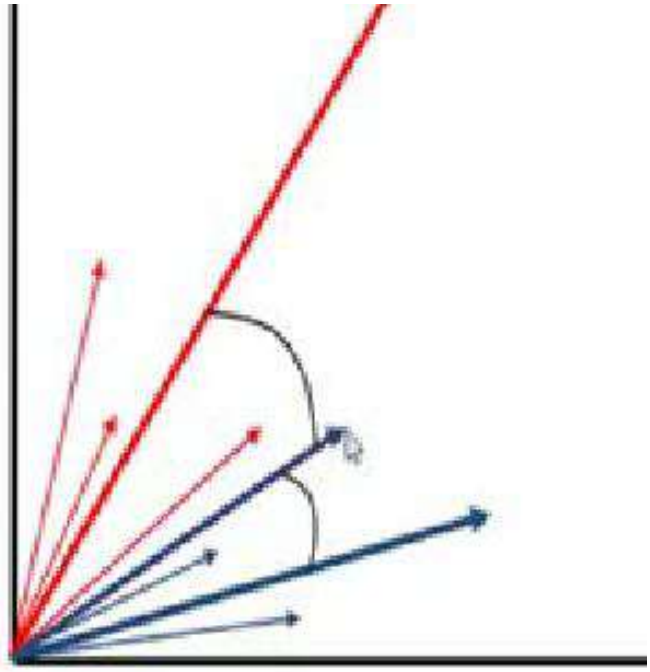
Rocchio Algorithm

- Relevance feedback methods can be adapted for text categorization
    - As noted before, relevance feedback can be viewed as 2-class classification
    - **Relevant** vs. non-relevant documents
- Use standard tf-idf weighted vectors to represent text documents
- For training documents in each category, compute a prototype vector by summing the vectors of the training documents in the category.
    - Prototype : centroid of members of class
- Assign test documents to the category with the closest prototype vector based on cosine similarity



Rocchio Algorithm

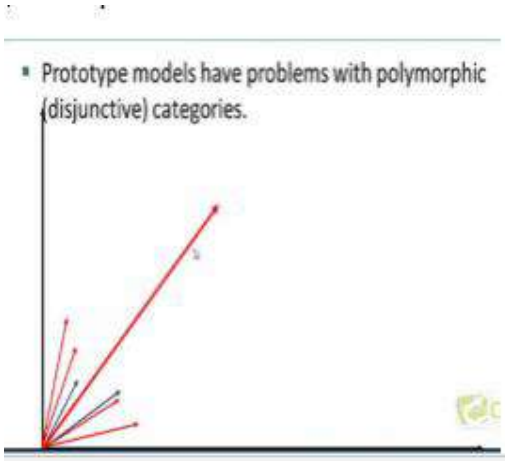
- There may be many more red vectors along y-axis, and they will drift the RED centroid towards y-axis. Now an awkward Red documents may be near the blue centroid.

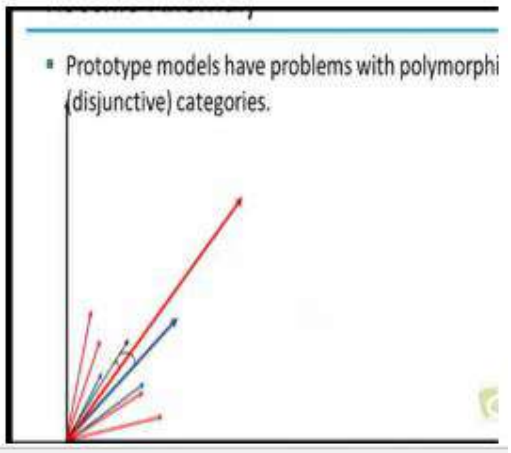


Rocchio classification

- Little used outside text classification

- It has been used quite effectively for text classification
- But in general worse than Naïve Bayes
- Again, cheap to train and test documents

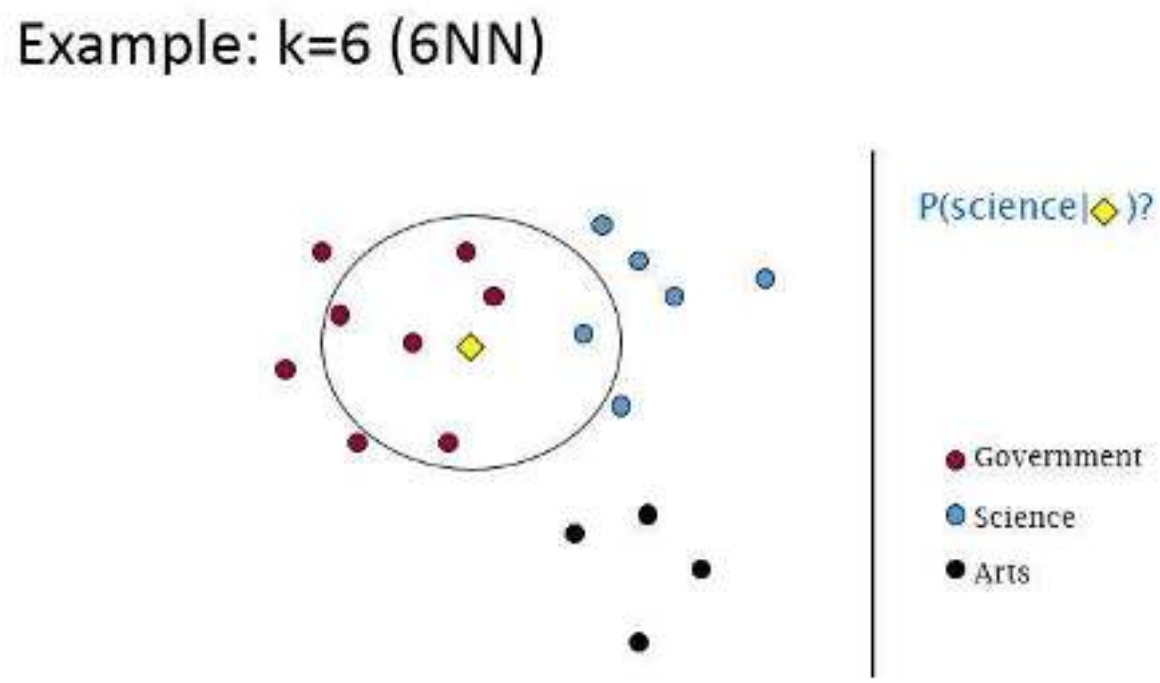




Rocchio classification

- Rocchio forms a simple representative for each class: the centroid/prototype
- Classification: nearest prototype/centroid
- It does not guarantee that classifications are consistent with the given training data

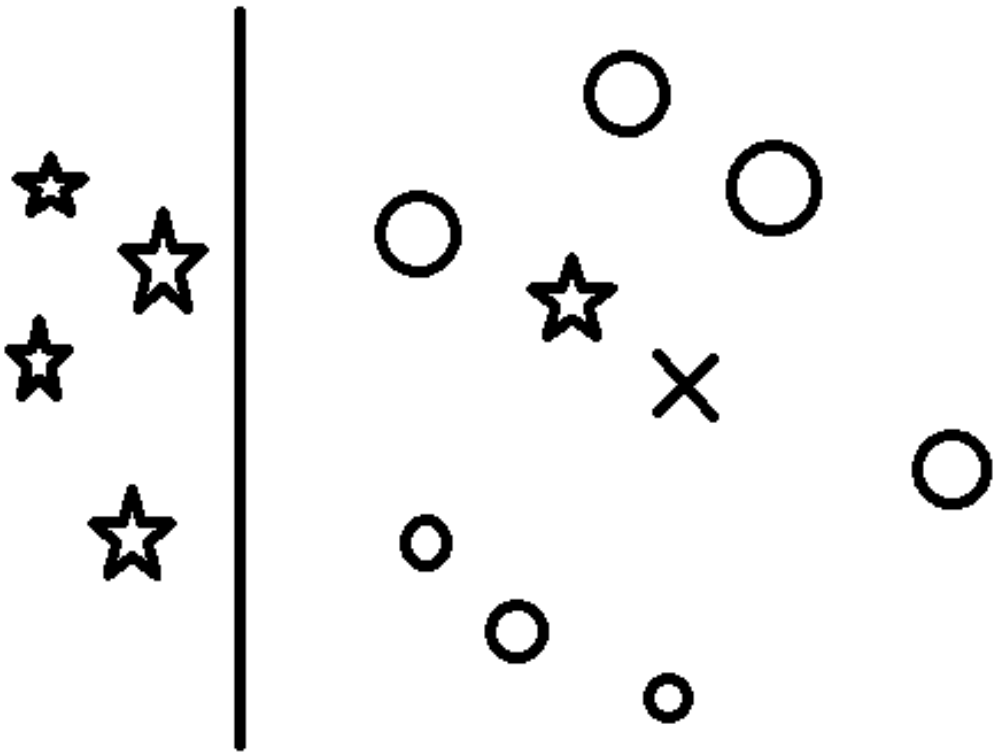
k Nearest Neighbor Classification



Nearest-Neighbor Learning

- Learning: just store the labeled training examples *D*
- Testing instance *x (under 1NN)*:
    - Compute similarity between *x* and all examples in *D*.
    - Assign *x* the category of the most similar example in *D*.
- Does not compute anything beyond storing the examples
- Also called:
    - Case-based learning (remembering every single example of each class)
    - Memory-based learning (memorizing every instance of training set)
    - Lazy learning
- Rationale of kNN: contiguity hypothesis (docs which are near to a given input doc will decide its class)

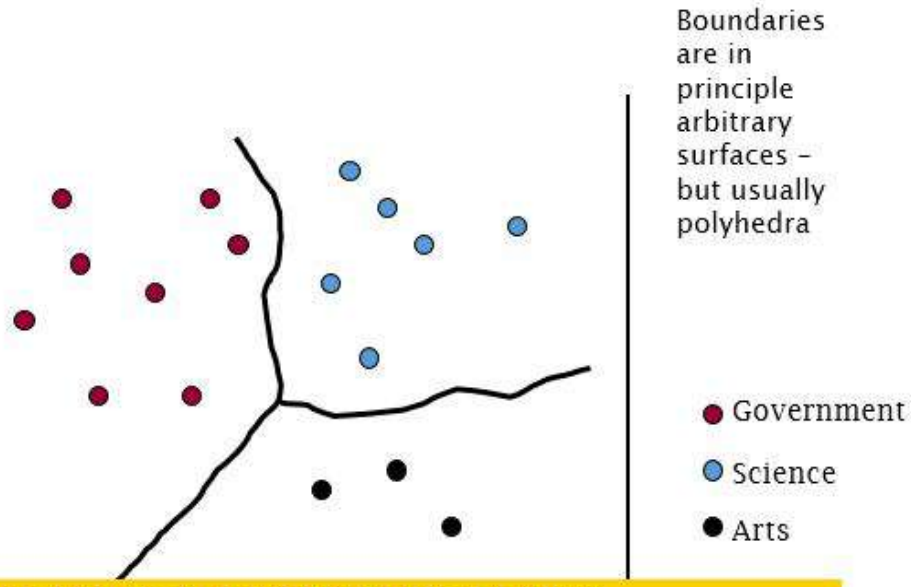
Nearest-Neighbor



k Nearest Neighbor

- Using only the closest example (1NN) subject to errors due to:
    - A single atypical example.
    - Noise (i.e., an error) in the category label of a single training example.
- More robust: find the *k* examples and return the majority category of these *k*
- *k* is typically odd to avoid ties; 3 and 5 are most common
- Assign weight (relevance) of neighbors to decide.

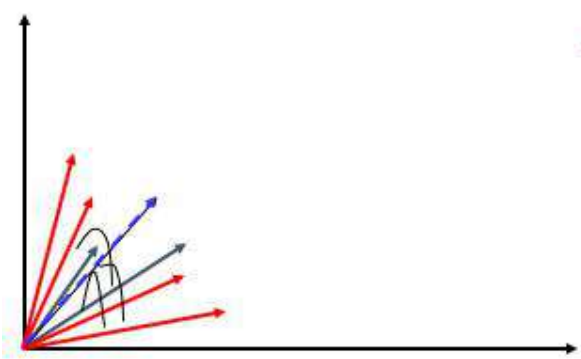
kNN decision boundaries

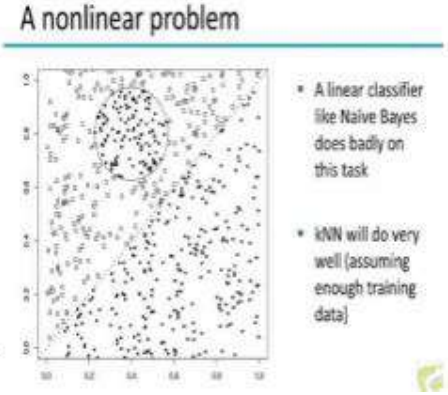


kNN gives locally defined decision boundaries between classes – far away points do not influence each classification decision (unlike in Naïve Bayes, Rocchio, etc.)

3 Nearest Neighbor vs. Rocchio

- Nearest Neighbor tends to handle polymorphic categories better than Rocchio/NB.





kNN: Discussion

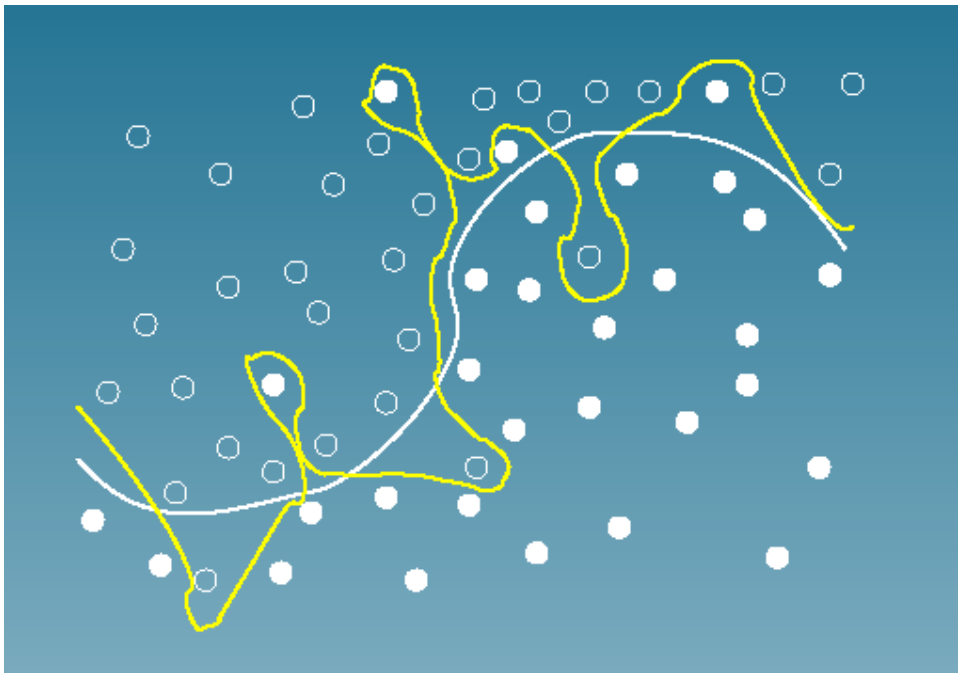
- No feature selection necessary
- No training necessary

- Scales well with large number of classes
    - Don't need to train *n* classifiers for *n* classes
- May be expensive at test time
- In most cases it's more accurate than NB or Rocchio

Evaluating Categorization

- Evaluation must be done on test data that are independent of the training data
    - Sometimes use cross-validation (averaging results over multiple training and test splits of the overall data)
- Easy to get good performance on a test set that was available to the learner during training (e.g., just memorize the test set)

Bias vs. variance: Choosing the correct model capacity



Lecture # 44

- Recommender Systems

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

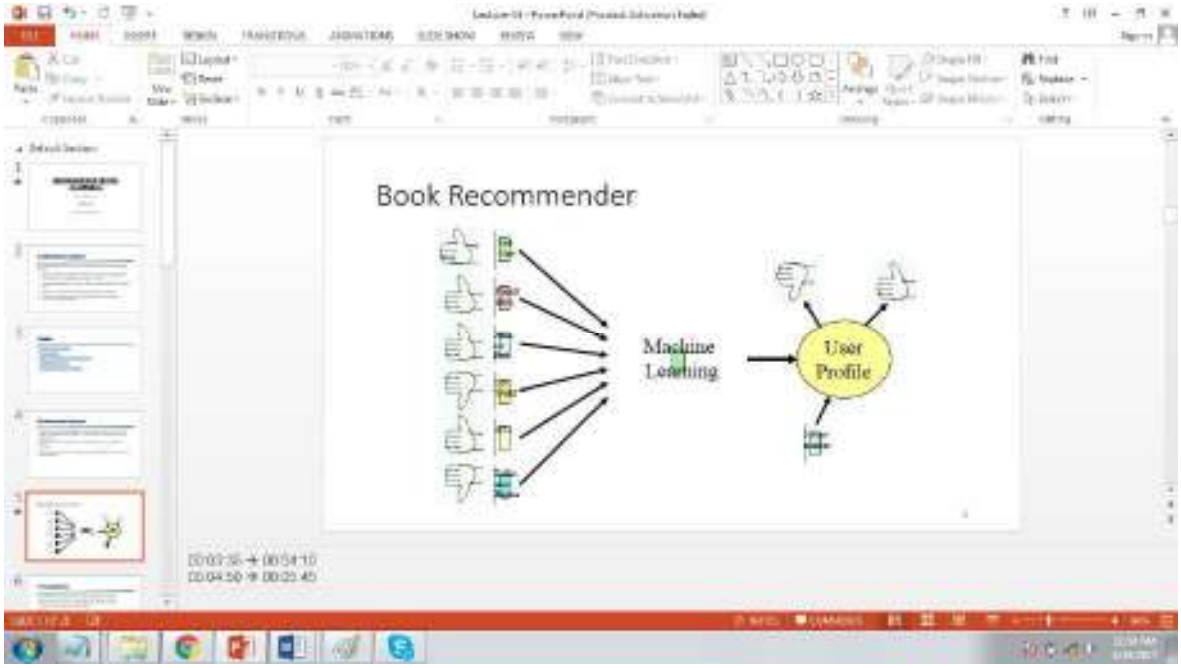
Outline

- Recommender Systems
- Personalization
- Basic Types of Recommender Systems
- Collaborative Filtering
- Content-Based Recommending

Recommender Systems

- Systems for recommending items (e.g. books, movies, CD's, web pages, newsgroup messages) to users based on examples of their preferences.
- Many on-line stores provide recommendations (e.g. Amazon, CDNow).
- Recommenders have been shown to substantially increase sales at on-line stores.

Book Recommender



Personalization

- Recommenders are instances of personalization software.
- Personalization concerns adapting to the individual needs, interests, and preferences of each user.
- Includes:
    - Recommending
    - Filtering
    - Predicting (e.g. form or calendar appt. completion)
- From a business perspective, it is viewed as part of Customer Relationship Management (CRM).

Basic Types of Recommender Systems

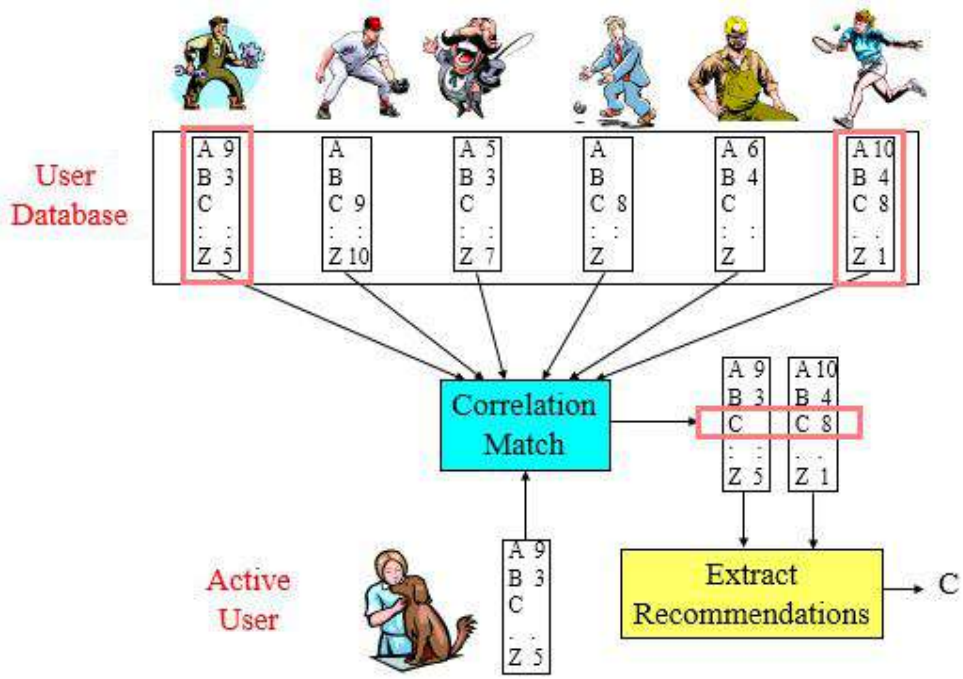
- There are two basic approaches to recommending:
    - Collaborative Filtering (a.k.a. social filtering)
    - Content-based

Collaborative Filtering

- Maintain a database of many users' ratings of a variety of items.
- For a given user, find other similar users whose ratings strongly correlate with the current user.
- Recommend items rated highly by these similar users, but not rated by the current user.

- Almost all existing commercial recommenders use this approach (e.g. Amazon).

Collaborative Filtering



Collaborative Filtering Method

- Weight all users with respect to similarity with the active user.
- Select a subset of the users (*neighbors*) to use as predictors.
- Normalize ratings and compute a prediction from a weighted combination of the selected neighbors' ratings.
- Present items with highest predicted ratings as recommendations.

Similarity Weighting

- Typically use Pearson correlation coefficient between ratings for active user, *a*, and another user, *u*.

$$c_{a,u} = \frac{\mathrm{covar}(r_a, r_u)}{\sigma_{r_a}\sigma_{r_u}}$$

r_a and r_u are the ratings vectors for the *m* items rated by **both** *a* and *u*

Covariance and Standard Deviation

- Covariance:

$$\mathrm{covar}(r_a, r_u) = \frac{\sum_{i=1}^{m} (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{m}$$

$$\bar{r}_x = \frac{\sum_{i=1}^{m} r_{x,i}}{m}$$

- $r_{x,y}$ is user *x*'s rating for item *y*
- Standard Deviation:

$$\sigma_{r_x} = \sqrt{\frac{\sum_{i=1}^{m} (r_{x,i} - \bar{r}_x)^2}{m}}$$

Significance Weighting

- Important not to trust correlations based on very few co-rated items.
- Include *significance weights*, $s_{a,u}$, based on number of co-rated items, *m*.

$$w_{a,u} = s_{a,u} c_{a,u}$$

$$s_{a,u} = \left\{ \begin{array}{l} 1 \text{ if } m > 50 \\ \frac{m}{50} \text{ if } m \leq 50 \end{array} \right\}$$

Neighbor Selection

- For a given active user, *a*, select correlated users to serve as source of predictions.
- Standard approach is to use the most similar *n* users, *u*, based on similarity weights, $w_{a,u}$

- Alternate approach is to include all users whose similarity weight is above a given threshold.

Rating Prediction

- Predict a rating, $p_{a,i}$, for each item *i*, for active user, *a*, by using the *n* selected neighbor users, $u \in \{1,2,\ldots n\}$.
- To account for users different ratings levels, base predictions on *differences* from a user's *average* rating.
- Weight users' ratings contribution by their similarity to the active user.

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u=1}^{n} w_{a,u}(r_{u,i} - \bar{r}_u)}{\sum_{u=1}^{n} w_{a,u}}$$

Problems with Collaborative Filtering

- **Cold Start**: There needs to be enough other users already in the system to find a match.
- **Sparsity**: If there are many items to be recommended, even if there are many users, the user/ratings matrix is sparse, and it is hard to find users that have rated the same items.
- **First Rater**: Cannot recommend an item that has not been previously rated.
    - New items
    - Esoteric items (unique)
- **Popularity Bias**: Cannot recommend items to someone with unique tastes.
    - Tends to recommend popular items.

Content-Based Recommending

- Recommendations are based on information on the content of items rather than on other users' opinions.

- Uses a machine learning algorithm to induce a profile of the users preferences from examples based on a featural description of content.
- Some previous applications:
    - Newsweeder (Lang, 1995)
    - Syskill and Webert (Pazzani et al., 1996)

Advantages of Content-Based Approach

- No need for data on other users.
    - No cold-start or sparsity problems.
- Able to recommend to users with unique tastes.
- Able to recommend new and unpopular items
    - No first-rater problem.
- Can provide explanations of recommended items by listing content-features that caused an item to be recommended.

Disadvantages of Content-Based Method

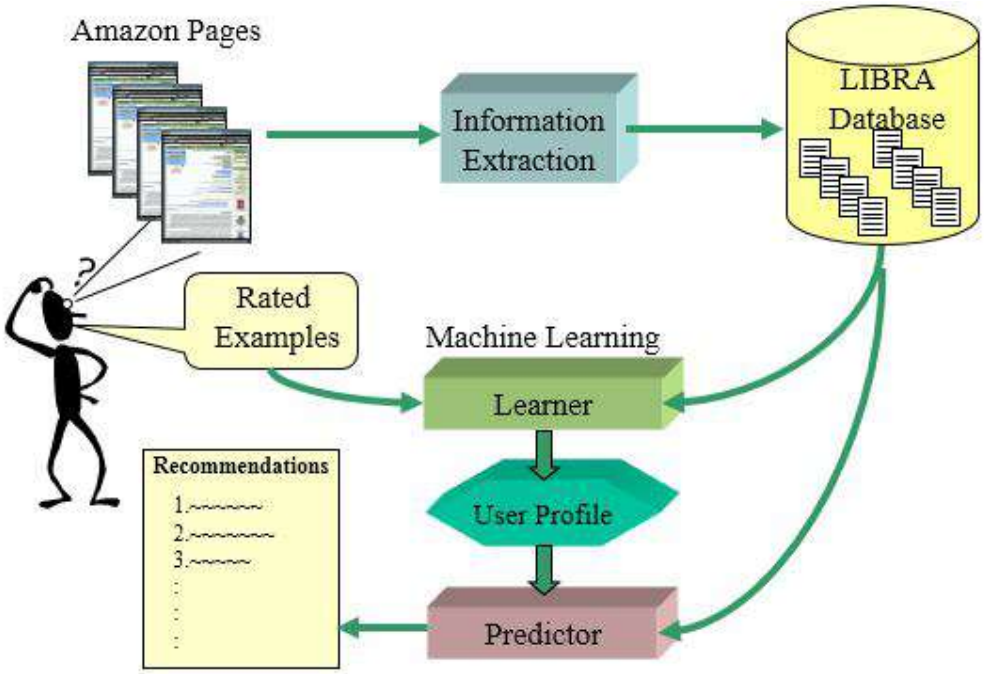
- Requires content that can be encoded as meaningful features.
- Users' tastes must be represented as a learnable function of these content features.
- Unable to exploit quality judgments of other users.
    - Unless these are somehow included in the content features.

LIBRA

Learning Intelligent Book Recommending Agent

- Content-based recommender for books using information about titles extracted from Amazon.
- Uses information extraction from the web to organize text into fields:
    - Author
    - Title
    - Editorial Reviews
    - Customer Comments
    - Subject terms
    - Related authors
    - Related titles

LIBRA System



Lecture # 45

- Final Notes on Information Retrieval

ACKNOWLEDGEMENTS

The presentation of this lecture has been taken from the following sources

1. "Introduction to information retrieval" by Prabhakar Raghavan, Christopher D. Manning, and Hinrich Schütze
2. "Managing gigabytes" by Ian H. Witten, Alistair Moffat, Timothy C. Bell
3. "Modern information retrieval" by Baeza-Yates Ricardo,
4. "Web Information Retrieval" by Stefano Ceri, Alessandro Bozzon, Marco Brambilla

Outline

- Topics that we covered
- Database Management Research

Topics that we covered

- IR Models

- Boolean / Vector Space / Probability IR
- IR System Implementation
  - Inverted Index (different levels)
  - Naïve Implementation – Scalable realistic imp
  - Optimizations (query processing, index building: compression)
- Types of Queries and data handling
- Deep Web Searching (Search computing)
- Web Based IR : Page Rank, Crawling
- Classification/ Clustering
- Recommender Systems

Database Management Research

How to improve web search and surfing

- **Personalization** (User Preferences, localization)
- **Social Data**
  - **Leverage community interaction** to create refine content
  - Experts, friends, sub-communities of shared interests
- **Collaborative Working**
  - **Wisdom of the crowd** (page rank, reviews, feedbacks…) (Crowd Sourcing)
  - **Collaborative editing** (Wikipedia); Collaborative Searching (crowd search)
  - **Harnessing the collaborative power** (Amazon Mechanical Turk)
- **Data Quality**
  - **Provenance**/Lineage/Source; **Confidence** on the source; **Correlation** (did we agree with source before) What is the mileage of my Honda Civic (so many sites..)

Database Management Research

- **Data Extraction**
  - Question answering relevant information
  - Knowledge vs Reasoning Engines (google vs wolfram alpha)
  - Domain Specific knowledge
- **Cross Lingual**
  - **Document Summarization**
  - **Plagiarism Detection**
- **Large-scale Data Management**
  - Scalability
    - Big Data Management;