# Chapter 5: Information Retrieval and Web Search

An introduction

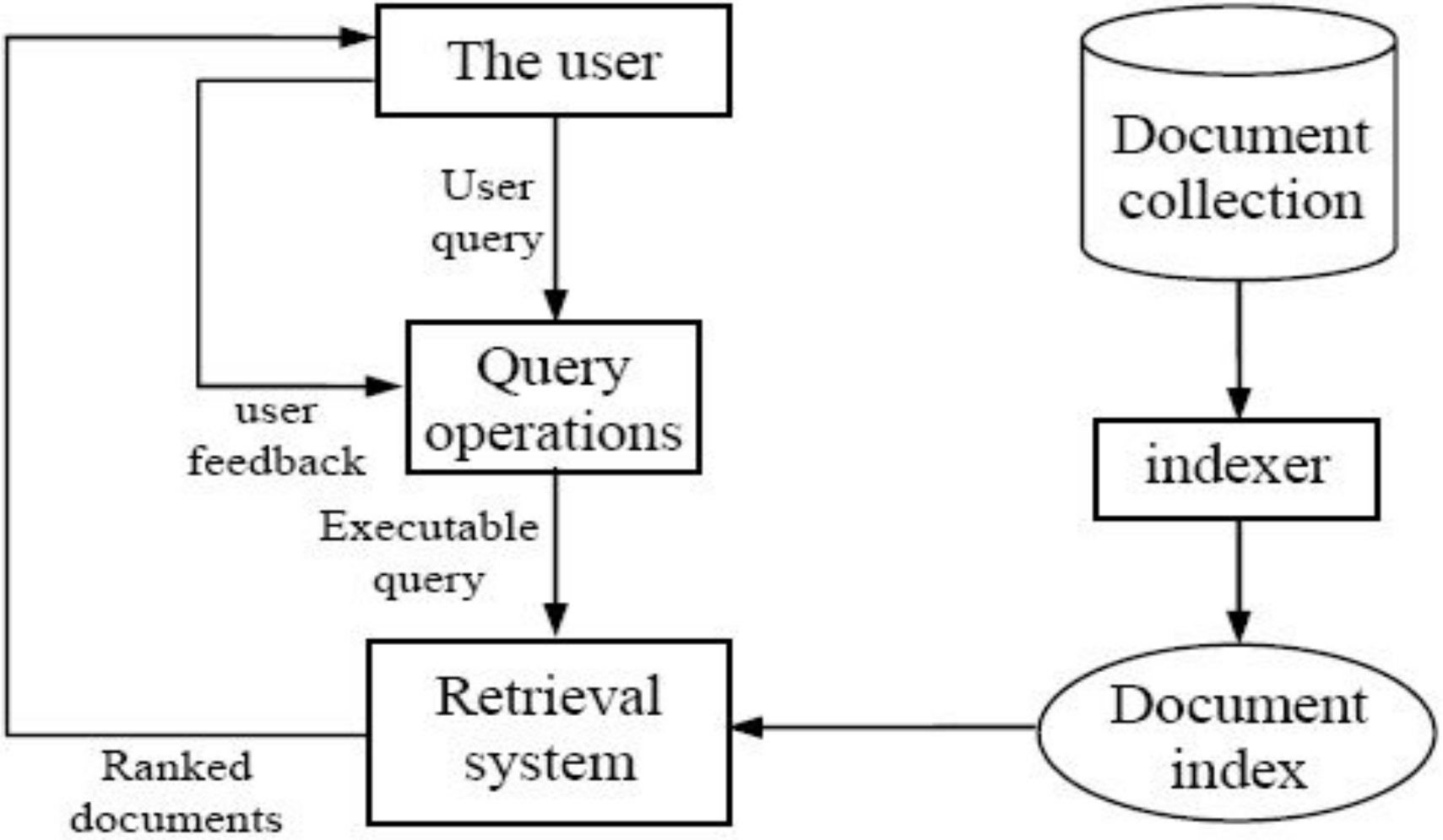
# Introduction

- Text mining refers to data mining using text documents as data.
- Most text mining tasks use Information Retrieval (IR) methods to pre-process text documents.
- These methods are quite different from traditional data pre-processing methods used for relational tables.
- Web search also has its root in IR.

# Information Retrieval (IR)

- Conceptually, IR is the study of finding needed information. I.e., IR helps users find information that matches their information needs.
  - Expressed as queries
- Historically, IR is about document retrieval, emphasizing document as the basic unit.
  - Finding documents relevant to user queries
- Technically, IR studies the acquisition, organization, storage, retrieval, and distribution of information.

# IR architecture



# IR System Architecture

- The user with information need issues a query (**user query)**to the **retrieval system through the query operations module.**
- The retrieval module uses the document index to retrieve those documents that contain some query terms (such documents are likely to be relevant to the query),
- It compute relevance scores for them, and then rank the retrieved documents according to the scores.
- The ranked documents are then presented to the user.
- The document collection is also called the text database, which is indexed by the indexer for efficient retrieval.

# IR queries

- Keyword queries
- Boolean queries (using AND, OR, NOT)
- Phrase queries
- Proximity queries
- Full document queries
- Natural language questions

# Queries

- **Keyword queries: The user expresses his/her information needs with a** list of (at least one) keywords (or **terms) aiming to find documents that** contain some (at least one) or all the query terms.
- The terms in the list are assumed to be connected with a "soft" version of the logical AND.
  - if one is interested in finding information about Web mining, one may issue the query 'Web mining' to an IR or search engine system. 'Web mining' is retreated as 'Web AND mining'.

# Queries

- **Boolean queries: The user can use Boolean operators, AND, OR, and NOT** to construct complex queries.
- Thus, such queries consist of terms and Boolean operators.
  - For example, 'data OR Web' is a Boolean query, which requests documents that contain the word 'data' or 'Web.
- A page is returned for a Boolean query if the query is logically true in the page (i.e., **exact match).**

# Queries

- **Phrase queries: Such a query consists of a sequence of words that** makes up a phrase.
- Each returned document must contain at least one instance of the phrase.
- In a search engine, a phrase query is normally enclosed with double quotes.
- For example, one can issue the following phrase query (including the double quotes), “Web mining techniques and applications” to find documents that contain the exact phrase.

# Queries

- **Proximity queries: The proximity query is a relaxed version of the** phrase query and can be a combination of terms and phrases.
- Proximity queries seek the query terms within close proximity to each other.
- The closeness is used as a factor in ranking the returned documents or pages.
- For example, a document that contains all query terms close together is considered more relevant than a page in which the query terms are far apart.

# Queries

- **Full document queries: When the query is a full document, the user** wants to find other documents that are similar to the query document.
- Some search engines (e.g., Google) allow the user to issue such a query by providing the URL of a query page.

# Queries

- **Natural language questions: This is the most complex case, and also**the ideal case.
- The user expresses his/her information need as a natural language question.
- This is an active research area, called **question answering.**

# Information retrieval models

- An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined.
- Main models:
  - Boolean model
  - Vector space model
  - Statistical language model
  - Probabilistic Model

# Information Retrieval Model

- Each document or query is treated as a **"bag" of words** or **terms**. Word sequence is not considered.
- Given a collection of documents $D$, let $V = \{t_1, t_2, ..., t_{|V|}\}$ be the set of distinctive words/terms in the collection. $V$ is called the vocabulary.
- A weight $w_{ij} > 0$ is associated with each term $t_i$ of a document $\mathbf{d}_j \in D$. For a term that does not appear in document $\mathbf{d}_j$, $w_{ij} = 0$.
- Each document $\mathbf{d}_j$ can be represented with a term vector

  $\mathbf{d}_j = (w_{1j}, w_{2j}, ..., w_{|V|j})$,

# Boolean Model

**Document Representation:**

- **In the Boolean model, documents and queries** are represented as sets of terms.
- That is, each term is only considered present or absent in a document
- The weight *wij (∈ {0, 1}) of term* $\boldsymbol{t_i}$ *in document* $\boldsymbol{d_j}$ ***is 1 if*** $\boldsymbol{t_i}$ *appears in document* $\boldsymbol{d_j}$***, and 0 otherwise, i.e.,***

$$W_{ij} = 1 \quad \text{if } t_i \text{ appears in } d_j$$
$$= 0 \quad \text{otherwise}$$

# Boolean model

- **Query Representation**

  Query terms are combined logically using the Boolean operators **AND**, **OR**, and **NOT.**
  - E.g., ((*data* AND *mining*) AND (NOT *text*))
- **Document Retrieval**
  - Given a Boolean query, the system retrieves every document that makes the query logically true.
  - Called **exact match**.
- The retrieval results are usually quite poor because
  - Term frequency is not considered.
  - There is no notion of partial match
  - No ranking of the retrieved documents.

# Vector space model

Document Representation

- Documents are also treated as a "bag" of words or terms.
- Each document is represented as a weight vector.
- However, the term weights are no longer 0 or 1. Each term weight is computed based on some variations of TF or TF-IDF scheme.
- Term Frequency (TF) Scheme: The weight of a term $t_i$ in document $\mathbf{d}_j$ is the number of times that $t_i$ appears in $\mathbf{d}_j$, denoted by $f_{ij}$. Normalization may also be applied.

# TF-IDF term weighting scheme

- The most well known weighting scheme
  - TF: still **term frequency**
  - IDF: **inverse document frequency**.

  $N$: total number of docs

  $df_i$: the number of docs that $t_i$ appears atleast once
- The final TF-IDF term weight is:

$$tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\}}$$

$$idf_i = \log \frac{N}{df_i}$$

$$w_{ij} = tf_{ij} \times idf_i.$$

# Vector Space Model

- Query **q** is represented in the same way or slightly differently

$$w_{iq} = \left(0.5 + \frac{0.5 f_{iq}}{\max\{f_{1q}, f_{2q}, \ldots, f_{|V|q}\}}\right) \times \log \frac{N}{df_i}.$$

# Retrieval in vector space model

- Vector Space Model does not make a binary decision on whether a document is relevant to a given query
- The documents are ranked according to their degrees of relevance to the query.
- Relevance of $\mathbf{d}_j$ to $\mathbf{q}$: Compare the similarity of query $\mathbf{q}$ and document $\mathbf{d}_j$.
- Cosine similarity (the cosine of the angle between the two vectors)

$$cosine(\mathbf{d}_j, \mathbf{q}) = \frac{\langle \mathbf{d}_j \bullet \mathbf{q} \rangle}{\| \mathbf{d}_j \| \times \| \mathbf{q} \|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

# An Example

- A document space is defined by three terms:
  - hardware, software, users
  - the vocabulary
- A set of documents are defined as:
  - A1=(1, 0, 0), A2=(0, 1, 0), A3=(0, 0, 1)
  - A4=(1, 1, 0), A5=(1, 0, 1), A6=(0, 1, 1)
  - A7=(1, 1, 1) A8=(1, 0, 1). A9=(0, 1, 1)
- If the Query is “hardware and software”
- what documents should be retrieved?

# An Example (cont.)

- In Boolean query matching:
  - document A4, A7 will be retrieved ("AND")
  - retrieved: A1, A2, A4, A5, A6, A7, A8, A9 ("OR")
- In similarity matching (cosine):
  - q=(1, 1, 0)
  - S(q, A1)=0.71, S(q, A2)=0.71, S(q, A3)=0
  - S(q, A4)=1, S(q, A5)=0.5, S(q, A6)=0.5
  - S(q, A7)=0.82, S(q, A8)=0.5, S(q, A9)=0.5
  - Document retrieved set (with ranking)=
    - {A4, A7, A1, A2, A5, A6, A8, A9}

# Okapi relevance method

- Another way to assess the degree of relevance is to directly compute a relevance score for each document to the query.
- The **Okapi** method and its variations are popular techniques in this setting.

The Okapi relevance score of a document $d_j$ for a query $q$ is:

$$okapi(d_j, q) = \sum_{t_i \in q, d_j} \ln \frac{N - df_i + 0.5}{df_i + 0.5} \times \frac{(k_1 + 1) f_{ij}}{k_1 (1 - b + b \frac{dl_j}{avdl}) + f_{ij}} \times \frac{(k_2 + 1) f_{iq}}{k_2 + f_{iq}},$$

where $k_1$ (between 1.0-2.0), $b$ (usually 0.75) and $k_2$ (between 1-1000)

# Statistical Language Model

- Statistical language models (or simply **language models) are based on** probability and have foundations in statistical theory
- It first estimates a language model for each document
- Then ranks documents by the likelihood of the query give

# Statistical Language Model

- Let the query *q be a sequence of terms,*

  *q = q1q2…qm and the document* collection *D be a set of documents,*

  *D = {d1, d2, …, dN}*

- In the language modeling approach, we consider the probability of a query *q as being* "generated" by a probabilistic model based on a document *dj, i.e., Pr(q|dj)*

- To rank documents in retrieval, estimate the posterior probability Pr(*dj*|*q)*

# Statistical Language Model

- By Bayes Rule

$$\Pr(d_j \mid q) = \frac{\Pr(q \mid d_j)\Pr(d_j)}{\Pr(q)}$$

- For ranking, Pr(*q*) *is not needed as it is the same for every document.* Pr(*dj*) *is usually considered uniform and thus will not affect ranking. We* only need to compute Pr(*q*|*dj*).

Based on the multinomial distribution and the unigram model,

$$\Pr(q = q_1 q_2 \ldots q_m \mid d_j) = \prod_{i=1}^{m} \Pr(q_i \mid d_j) = \prod_{i=1}^{|V|} \Pr(t_i \mid d_j)^{f_{iq}},$$

# Statistical Language Model

- where $f_{iq}$ *is the number of times that term ti occurs in q, and the retrieval problem is reduced to estimating Pr(ti|dj),*
- It can be the relative frequency,

$$\Pr(t_i \mid d_j) = \frac{f_{ij}}{|d_j|}.$$

- $f_{ij}$ *is the number of times that term* $t_i$ *occurs in document* $d_j$. $|d_j|$ denotes the total number of words in *dj.*

# Statistical Language Model

- One problem with this estimation is that a term that does not appear in *dj has the probability of 0, which underestimates the probability* of the unseen term in the document.
- This situation is similar to text classification using the naïve Bayesian model
- A non-zero probability is typically assigned to each unseen term in the document,which is called **smoothing.**
- **Smoothing adjusts the estimates of** probabilities to produce more accurate probabilities.
- The name smoothing comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and high probabilities downward.

# Relevance feedback

- Relevance feedback is one of the techniques for improving retrieval effectiveness. The steps:
  - the user first identifies some relevant ($D_r$) and irrelevant documents ($D_{ir}$) in the initial list of retrieved documents
  - the system expands the query **q** by extracting some additional terms from the sample relevant and irrelevant documents to produce $\mathbf{q}_e$
  - Perform a second round of retrieval.
- Rocchio method ($\alpha$, $\beta$ and $\gamma$ are parameters)

$$\mathbf{q}_e = \alpha\mathbf{q} + \frac{\beta}{|D_r|}\sum_{\mathbf{d}_r \in D_r}\mathbf{d}_r - \frac{\gamma}{|D_{ir}|}\sum_{\mathbf{d}_{ir} \in D_{ir}}\mathbf{d}_{ir}$$

# Rocchio text classifier

- In fact, a variation of the Rocchio method above, called the **Rocchio classification** method, can be used to improve retrieval effectiveness too
  - so are other machine learning methods. Why?
- Rocchio classifier is constructed by producing a prototype vector $\mathbf{c}_i$ for each class $i$ (*relevant* or *irrelevant* in this case):

$$\mathbf{c}_i = \frac{\alpha}{|D_i|}\sum_{\mathbf{d}\in D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|} - \frac{\beta}{|D - D_i|}\sum_{\mathbf{d}\in D - D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|}$$

- In classification, cosine is used.

# Evaluation

- Let the collection of documents in the database be *D, and the total* number of documents in *D be N.*
- *Given a user query* ***q, the retrieval*** algorithm first computes relevance scores for all documents in *D*
- *Then* produce a ranking *Rq of the documents based on the relevance scores, i.e.*

$$R_q : < \mathbf{d}_1^q, \mathbf{d}_2^q, ..., \mathbf{d}_N^q >, \quad (16)$$

where $\mathbf{d}_1^q \in D$ is the most relevant document to query $\mathbf{q}$ and $\mathbf{d}_N^q \in D$ is the most irrelevant document to query $\mathbf{q}$.

Let $D_q$ ($\subseteq D$) be the set of actual relevant documents of query $\mathbf{q}$ in $D$. We can compute the precision and recall values at each $\mathbf{d}_i^q$ in the ranking.

# Evaluation: Precision and Recall

- Given a query:
  - Are all retrieved documents relevant?
  - Have all the relevant documents been retrieved?
- Measures for system performance:
  - **The first question is about the precision of the search**

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

  - **The second is about the completeness (recall) of the search.**

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

# Precision and Recall

**Recall** at rank position $i$ or document $\mathbf{d}_i^q$ (denoted by $r(i)$) is the fraction of relevant documents from $\mathbf{d}_1^q$ to $\mathbf{d}_i^q$ in $R_q$. Let the number of relevant documents from $\mathbf{d}_1^q$ to $\mathbf{d}_i^q$ in $R_q$ be $s_i$ ($\leq |D_q|$) ($|D_q|$ is the size of $D_q$).

$$r(i) = \frac{s_i}{|D_q|}. \quad (17)$$

**Precision** at rank position $i$ or document $\mathbf{d}_i^q$ (denoted by $p(i)$) is the fraction of documents from $\mathbf{d}_1^q$ to $\mathbf{d}_i^q$ in $R_q$ that are relevant:

$$p(i) = \frac{s_i}{i} \quad (18)$$

# Example

**Example 1:** We have a document collection $D$ with 20 documents. Given a query **q**, we know that eight documents are relevant to **q**. A retrieval algorithm produces the ranking (of all documents in $D$) shown in Fig. 6.3.

| Rank $i$ | +/– | $p(i)$ | $r(i)$ |
|---|---|---|---|
| 1 | + | 1/1 = 100% | 1/8 = 13% |
| 2 | + | 2/2 = 100% | 2/8 = 25% |
| 3 | + | 3/3 = 100% | 3/8 = 38% |
| 4 | – | 3/4 = 75% | 3/8 = 38% |
| 5 | + | 4/5 = 80% | 4/8 = 50% |
| 6 | – | 4/6 = 67% | 4/8 = 50% |
| 7 | + | 5/7 = 71% | 5/8 = 63% |
| 8 | – | 5/8 = 63% | 5/8 = 63% |
| 9 | + | 6/9 = 67% | 6/8 = 75% |
| 10 | + | 7/10 = 70% | 7/8 = 88% |
| 11 | – | 7/11 = 63% | 7/8 = 88% |
| 12 | – | 7/12 = 58% | 7/8 = 88% |
| 13 | + | 8/13 = 62% | 8/8 = 100% |
| 14 | – | 8/14 = 57% | 8/8 = 100% |
| 15 | – | 8/15 = 53% | 8/8 = 100% |
| 16 | – | 8/16 = 50% | 8/8 = 100% |
| 17 | – | 8/17 = 53% | 8/8 = 100% |
| 18 | – | 8/18 = 44% | 8/8 = 100% |
| 19 | – | 8/19 = 42% | 8/8 = 100% |
| 20 | – | 8/20 = 40% | 8/8 = 100% |

# Precision–Recall Curve

- **Based on the precision and recall values at each** rank position, we can draw a precision–recall curve
- *x-axis is the* recall and the *y-axis is the precision.*
- *Instead of using the precision and* recall at each rank position, the curve is commonly plotted using 11standard recall levels, 0%, 10%, 20%, …, 100%.

# Interpolation

- Interpolation is needed to obtain the precisions at these recall levels, which is done as follows:

Let $r_i$ be a recall level, $i \in \{0, 1, 2, \ldots, 10\}$, and $p(r_i)$ be the precision at the recall level $r_i$. $p(r_i)$ is computed with

$$p(r_i) = \max_{r_i \le r \le r_{10}} p(r) . \quad (21)$$

That is, to interpolate precision at a particular recall level $r_i$, we take the maximum precision of all recalls between level $r_i$ and level $r_{10}$.

# Precision-recall curve

**Example 2:** Following Example 1, we obtain the interpolated precisions at all 11 recall levels in the table of Fig. 6.4. The precision-recall curve is shown on the right.

| $i$ | $p(r_i)$ | $r_i$ |
|---|---|---|
| 0 | 100% | 0% |
| 1 | 100% | 10% |
| 2 | 100% | 20% |
| 3 | 100% | 30% |
| 4 | 80% | 40% |
| 5 | 80% | 50% |
| 6 | 71% | 60% |
| 7 | 70% | 70% |
| 8 | 70% | 80% |
| 9 | 62% | 90% |
| 10 | 62% | 100% |

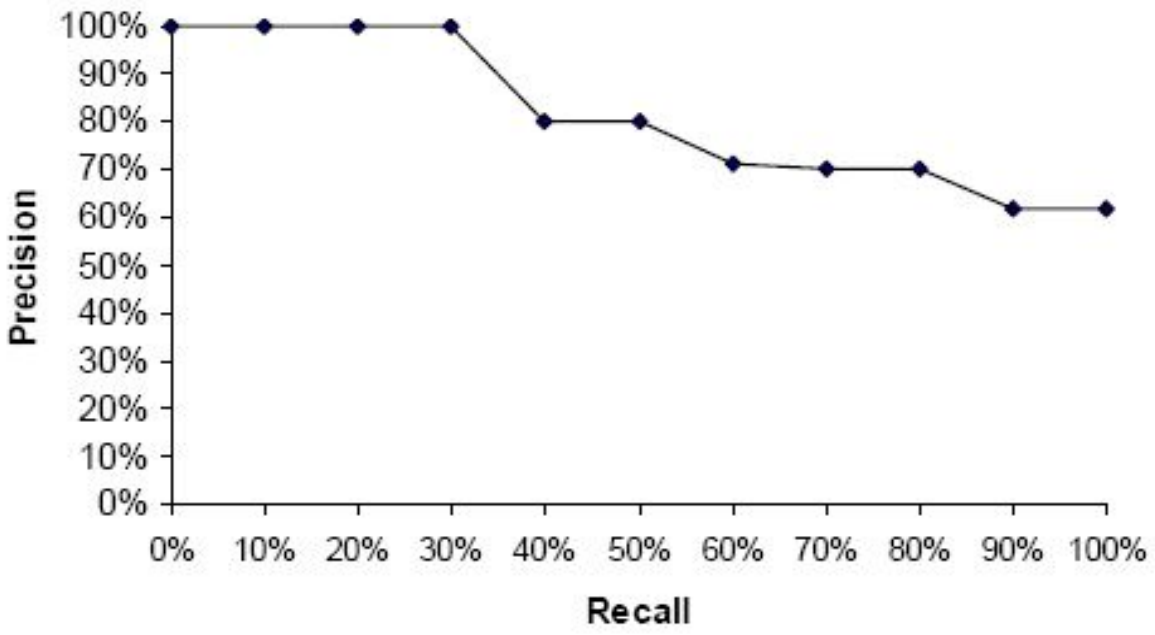



**Fig. 6.4.** The precision-recall curve

# Compare different retrieval algorithms

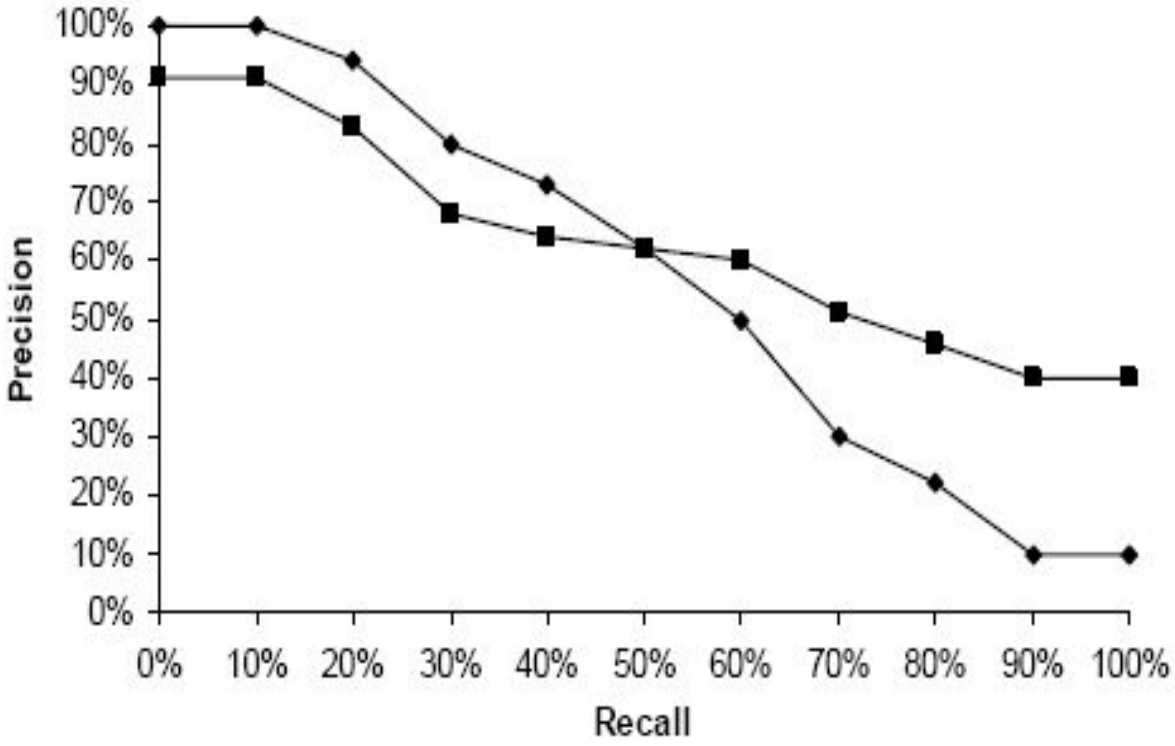



**Fig. 6.5.** Comparison of two retrieval algorithms based on their precision-recall curves

# Evaluation Using Multiple Queries

The overall precision (denoted by $\overline{p}(r_i)$) at each recall level $r_i$ is computed as the average of individual precisions at that recall level, i.e.,

$$\overline{p}(r_i) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} p_j(r_i), \qquad (22)$$

where $Q$ is the set of all queries and $p_j(r_i)$ is the precision of query $j$ at the recall level $r_i$.

# Rank precision

- Compute the precision values at some selected rank positions.
- Mainly used in Web search evaluation.
- For a Web search engine, we can compute precisions for the top 5, 10, 15, 20, 25 and 30 returned pages
- Recall does not make much sense for Web search because the user seldom looks at pages ranked below 30.
- Precision is critical, and it can be estimated for top ranked documents.

# F-Score

- We can compute the F-score at each rank position *i.*
- *F-score is the harmonic mean of precision and recall*

$$F(i) = \frac{2}{\frac{1}{r(i)} + \frac{1}{p(i)}} = \frac{2p(i)r(i)}{p(i) + r(i)}.$$

# Text pre-processing

- Word (term) extraction: easy
- Stopwords removal
- Stemming
- Frequency counts and computing TF-IDF term weights.

# Stopwords removal

- Many of the most frequently used words in English are useless in IR and text mining – these words are called *stop words*.
  - the, of, and, to, ….
  - Typically about 400 to 500 such words
  - For an application, an additional domain specific stopwords list may be constructed
- Why do we need to remove stopwords?
  - Reduce indexing (or data) file size
    - stopwords accounts 20-30% of total word counts.
  - Improve efficiency and effectiveness
    - stopwords are not useful for searching or text mining
    - they may also confuse the retrieval system.

# Stemming

- Techniques used to find out the root/stem of a word. E.g.,
  - user
  - users
  - used
  - using

  engineering
  engineered
  engineer
- stem: use engineer

**Usefulness:**

- improving effectiveness of IR and text mining
  - matching similar words
  - Mainly improve recall
- reducing indexing size
  - combing words with same roots may reduce indexing size as much as 40-50%.

# Basic stemming methods

Using a set of rules. E.g.,

- remove ending
  - if a word ends with a consonant other than s, followed by an s, then delete s.
  - if a word ends in es, drop the s.
  - if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th.
  - If a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter.
  - ……
- transform words
  - if a word ends with “ies” but not “eies” or “aies” then “ies --> y.”

# Frequency counts + TF-IDF

- Counts the number of times a word occurred in a document.
  - Using occurrence frequencies to indicate relative importance of a word in a document.
    - if a word appears often in a document, the document likely "deals with" subjects related to the word.
- Counts the number of documents in the collection that contains each word
- TF-IDF can be computed.

# Web Search as a huge IR system

- A Web crawler (robot) crawls the Web to collect all the pages.
- Servers establish a huge inverted indexing database and other indexing databases
- At query (search) time, search engines conduct different types of vector query matching.

# Inverted index

- The inverted index of a document collection is basically a data structure that
  - attaches each distinctive term with a list of all documents that contains the term.
- Thus, in retrieval, it takes constant time to
  - find the documents that contains a query term.
  - multiple query terms are also easy handle as we will see soon.

# An example

**Example 3:** We have three documents of $id_1$, $id_2$, and $id_3$:

$id_1$: Web mining is useful.
1 2 3 4

$id_2$: Usage mining applications.
1 2 3

$id_3$: Web structure mining studies the Web hyperlink structure.
1 2 3 4 5 6 7 8

| | |
|---|---|
| Applications: | $id_2$ |
| Hyperlink: | $id_3$ |
| Mining: | $id_1$, $id_2$, $id_3$ |
| Structure: | $id_3$ |
| Studies: | $id_3$ |
| Usage: | $id_2$ |
| Useful: | $id_1$ |
| Web: | $id_1$, $id_3$ |

(A)

| | |
|---|---|
| Applications: | <$id_2$, 1, [3]> |
| Hyperlink: | <$id_3$, 1, [7]> |
| Mining: | <$id_1$, 1, [2]>, <$id_2$, 1, [2]>, <$id_3$, 1, [3]> |
| Structure: | <$id_3$, 2, [2, 8]> |
| Studies: | <$id_3$, 1, [4]> |
| Usage: | <$id_2$, 1, [1]> |
| Useful: | <$id_1$, 1, [4]> |
| Web: | <$id_1$, 1, [1]>, <$id_3$, 2, [1, 6]> |

(B)

**Fig. 6.7.** Two inverted indices: a simple version and a more complex version

# Index construction

- Easy! See the example,

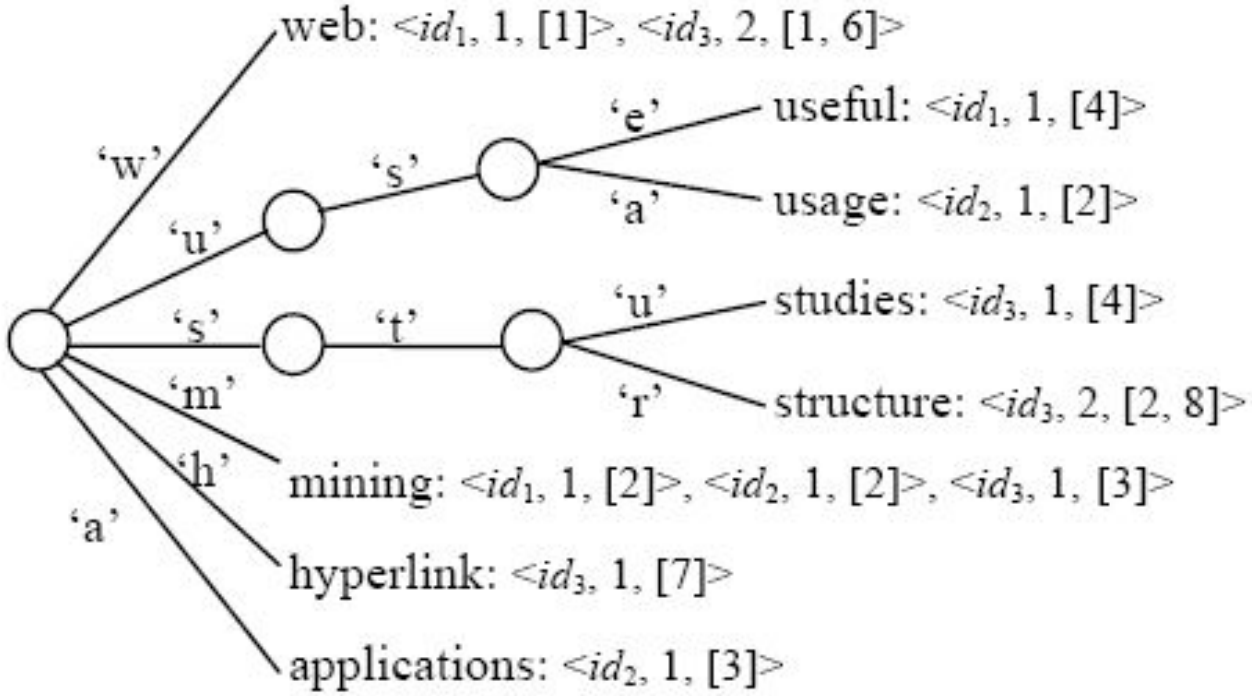



**Fig. 6.8.** The vocabulary trie and the inverted lists

# Search using inverted index

Given a query **q**, search has the following steps:

- Step 1 (**vocabulary search**): find each term/word in **q** in the inverted index.
- Step 2 (**results merging)**: Merge results to find documents that contain all or some of the words/terms in **q**.
- Step 3 (**Rank score computation**): To rank the resulting documents/pages, using,
  - content-based ranking
  - link-based ranking

# Different search engines

- The real differences among different search engines are
  - their index weighting schemes
    - Including location of terms, e.g., title, body, emphasized words, etc.
  - their query processing methods (e.g., query classification, expansion, etc)
  - their ranking algorithms
  - Few of these are published by any of the search engine companies. They are tightly guarded secrets.

# Summary

- We only give a VERY brief introduction to IR. There are a large number of other topics, e.g.,
  - Statistical language model
  - Latent semantic indexing (LSI and SVD).
  - (read an IR book or take an IR course)
- Many other interesting topics are not covered, e.g.,
  - Web search
    - Index compression
    - Ranking: combining contents and hyperlinks
  - Web page pre-processing
  - Combining multiple rankings and meta search
  - Web spamming
- Want to know more? Read the textbook