****

**Ambo University Woliso Campus**

**School of Technology and Informatics**

**Department of Information Technology**

Course title: - Information Retrieval and Storage

**Module Name: -** Information Management **Program: - Regular**

**Module code: - ITec-M3081 Target Group: -IT 3rd**

**Course Name: - information retrieval and storage Semester: - II**

**Course code: - ITec3081 Content: - Course Handout**

**Academic year: - 2020/2012 ECTS credits: - 5**

**Status of the course: - Instructor Name: - Husen Adem**

**Instructor Email: -** [igguumuude@gmail.com](mailto:igguumuude@gmail.com) **Phone Number: - +251925100878**

**Course Objectives**

***At the end of this course you will be able to****:*

* Understand the various Information Retrieval Systems and processes
* Know the retrieval model and evaluation of Information Retrieval Systems
* Understand the processes of information storage and retrieval
* Design, develop and evaluate information retrieval models
* Understand evaluation issues in IR
* Understand current issues in IR
* Try to retrieve information by using python
* Develop their own to information retrieval



Chapter 1:  
Introduction to Information Storage and Retrieval

Information Retrieval Systems?

* Document (Web page) retrieval in response to a query
  + Quite effective (at some things)
  + Commercially successful (some of them)
* But what goes on behind the scenes?
  + How do they work?
* Web search systems
* Lycos, Excite, Yahoo, Google, Live, Northern Light, Teoma, HotBot, Baidu, …
  + What happens beyond the Web?

Examples of IR systems

* Conventional (library catalog): Search by keyword, title, author, etc.
* Text-based (Lexis-Nexis, Google, FAST): Search by keywords. Limited search using queries in natural language.
* Multimedia (QBIC, WebSeek, SaFe): Search by visual appearance (shapes, colors, ).
* Question answering systems (AskJeeves, Answerbus): Search in (restricted) natural language
* Other:
  + Cross language information retrieval,
  + Music retrieval

**Information Retrieval**

* Information retrieval (IR) is the process of finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).
* Information is organized into (a large number of) documents
  + Large collections of documents from various sources: news articles, research papers, books, digital libraries, Web pages, etc.
  + **Example**: Web Search Engines like Google claim to index over 1 Trillion pages

**General Goal of Information Retrieval**

* To help users find useful information based on their information needs (with a minimum effort) despite
  + Increasing complexity of Information
  + Changing needs of user
* Provide immediate random access to the document collection.
* Retrieval systems, such as Google, Yahoo, are developed with this aim.

**Information Retrieval vs. Data Retrieval**

* Emphasis of IR is on the retrieval of information, rather than on the retrieval of data
* **Data retrieval**
  + Consists mainly of determining which documents contain a set of keywords in the user query (which is not enough to satisfy the user information need)
  + Aims at retrieving all objects that satisfy well defined semantics
  + a single erroneous object among a thousand retrieved objects implies failure
* **Information retrieval**
  + Is concerned with retrieving information about a subject or topic than retrieving data which satisfies a given query
  + semantics is frequently loose: the retrieved objects might be inaccurate
  + small errors are tolerated

**Information Retrieval vs. Data Retrieval**

* Example of data retrieval system is a relational database

|  |  |  |
| --- | --- | --- |
|  | **Data retrieval** | **Information Retrieval** |
| Data organization | Structured | Unstructured |
| Fields | Clear Semantics (ID, Name, age,) | No fields (other than text) |
| Query Language | Artificial (defined, SQL) | Free text (“natural language”), Boolean |
| Matching | Exact (results are *always* “correct”) | Partial match, best match |
| Query specification | Complete | Incomplete |
| Query specification | Complete | Incomplete |
| Items wanted | Matching | Relevant |
| Accuracy | 100% | < 50% |

**Why is IR so hard?**

* Traditionnel Information retrieval *(IR) System* attempt to find relevant documents to respond to a user’s request.
* **Information retrieval problem**: locating relevant documents based on user input, such as keywords or example documents
  + The real problem boils down to matching the language of the query to the language of the document.
  + Simply matching on words is a very brittle (no elasticity) approach. One word can have different semantic meanings. Consider: Take
  + “take a place at the table”
  + “take money to the bank”
  + “take a picture”

**More Problems with IR**

* You can’t even tell what part of speech a word has:
  + “I saw her duck”
  + A query that searches for “pictures of a duck” will find documents that contains:
    - “I saw her duck away from the ball falling from the sky”
* Proper Nouns often use regular old nouns
  + Consider a document with “a man named Abraham owned a Lincoln”
  + A word matching query for “Abraham Lincoln” may well find the above document.
* In the case of web, lack of well-defined data model for the web

**Basic Concepts in Information Retrieval:**

Effective retrieval of relevant information is directly affected by

(I) User Task and (ii) Logical View of documents adopted

**The User Task:**

two user task – retrieval and browsing



Retrieval

Browsing

DB

**USER**

**The User Task Retrieval**

* **It** is the process of retrieving information whereby the main objective is clearly defined from the onset of searching process.
* The user of a retrieval system has to translate his information need into a query in the language provided by the system.
* In this context (i.e. by specifying a set of words), the user searches for useful information executing a retrieval task
* English Language Statement:
* I want a book by J. K Rowling titled The Chamber of Secrets

**Browsing**

* **It** is the process of retrieving information, whereby the main objective is not clearly defined from the beginning and whose purpose might change during the interaction with the system.
* E.g. User might search for documents about ‘car racing’. Meanwhile he might find interesting documents about ‘car manufacturers. While reading about car manufacturers in Addis, he might turn his attention to a document providing ‘direction to Addis’, and from this to documents which cover ‘Tourism in Ethiopia’.
* In this context, user is said to be browsing in the collection and not searching, since a user may have an interest glancing around

**Logical View of Documents**

* Documents in a collection are frequently represented by a set of index terms or keywords
* Such keywords are mostly extracted directly from the text of the document
* These representative keywords provide a logical view of the document

Tokenization

stop words

stemming

Indexin**g**

Docs

Full text

Index terms

* Document representation viewed as a continuum, in which logical view of documents might shift from full text to index terms
* If full text:
  + Each word in the text is a keyword
  + Most complex form
  + Expensive
  + If full text is too large, the set of representative keywords can be reduced through transformation process called text operation
  + It reduces the complexity of the document representation and allow moving the logical view from that of a full text to a set of index terms

Structure of an IR System

* An Information Retrieval System serves as a bridge between the world of authors and the world of readers/users,
* That is, writers present a set of ideas in a document using a set of concepts. Then Users seek the IR system for relevant documents that satisfy their information need.

Black box

User

Documents

* The black box is the information retrieval system.
* To be effective in its attempt to satisfy information need of users, the IR system must ‘interpret’ the contents of documents in a collection and rank them according to their degree of relevance to the user query.
* Thus, the notion of relevance is at the centre of IR
* The primary goal of an IR system is to retrieve all the documents which are relevant to a user query while retrieving as few non-relevant documents as possible

**Typical IR Task**

* Given:
  + A corpus of textual natural-language documents.
  + A user query in the form of a textual string.
* Find:
  + A ranked set of documents that are relevant to the query.

**Typical IR System Architecture**



IR

System

Query String



1. Doc1

2. Doc2

3. Doc3

.

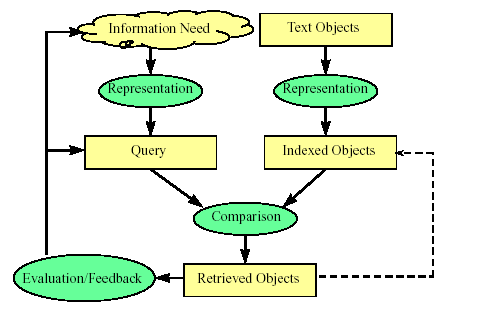
.

**Web Search System**

**What is Information Retrieval?**

* A good formal definition of information retrieval is given in Baeze-Yates & Riberio-Neto (1990, p1)
* “Information retrieval deals with representation, storage, organization of, and access to information items. The organization and access of information items should provide the user with **easy access to the information in which he is interested**”
* The definition incorporates all important features of a good information retrieval system
  + Representation
  + Storage
  + Organization
  + Access
* The focus is on the *user information need*

**Overview of the Retrieval process**



* It is necessary to define the text database before any of the retrieval processes are initiated
* This is usually done by the manager of the database and includes specifying the following
  + The documents to be used
  + The operations to be performed on the text
  + The text model to be used (the text structure and what elements can be retrieved)
* The text operations transform the original documents and the information needs and generate a logical view of them
* Once the logical view of the documents is defined, the database module builds an index of the text
  + An index is a critical data structure
  + It allows fast searching over large volumes of data
* Different index structures might be used, but the most popular one is the inverted file
* Given that the document database is indexed, the retrieval process can be initiated
* The user first specifies a user need which is then parsed and transformed by the same text operation applied to the text
* Next the query operations are applied before the actual query, which provides a system representation for the user need, is generated
* The query is then processed to obtain the retrieved documents
  + Before the retrieved documents are sent to the user, the retrieved documents are ranked according to the likelihood of relevance
* The user then examines the set of ranked documents in the search for useful information. Two choices for the user:
  + (I) reformulate query, run on entire collection or (ii) reformulate query, run on result set
* At this point, s/he might pinpoint a subset of the documents seen as definitely of interest and initiate a user feedback cycle
  + In such a cycle, the system uses the documents selected by the user to change the query formulation.
  + Hopefully, this modified query is a better representation of the real user need

**Detail view of the Retrieval Process**

User

Interface

Text Operations

Query Language & Operations

Indexing

Searching

Ranking

Index

Text

Query

User need

User feedback

Ranked docs

Retrieved docs

Logical view

logical view

Inverted file

DB manager

Module

Text Database

Text

**Issues that arise in IR**

* Text representation
  + what makes a “good” representation?
  + how is a representation generated from text?
  + what are retrievable objects and how are they organized?
* information needs representation
  + what is an appropriate query language?
  + how can interactive query formulation and refinement be supported?
* Comparing representations (to identify relevant documents)
  + What weighting scheme and similarity measure to be used?
  + what is a “good” model of retrieval?
* Evaluating effectiveness of retrieval
  + what are good metrics?
  + what constitutes a good experimental test bed?

**Focus in IR System Design**

Our focus during IR system design is:

* In improving performance effectiveness of the system
  + Effectiveness of the system is measured in terms of precision, recall, …
  + Stemming, stop words, weighting schemes, matching algorithms
* In improving performance efficiency
  + The concern here is storage space usage, access time, searching time, data transfer time …
  + Concern regarding space – time tradeoffs!!
  + Use Compression techniques, data/file structures, etc.

**Subsystems of an IR system**

* The two subsystems of an IR system:
  + Searching: is an online process of finding relevant documents in the index list as per user’s query
  + Indexing: is an offline process of organizing documents using keywords extracted from the collection
* Indexing and searching: are unavoidably connected
  + you cannot search what was not first indexed in some manner or other
  + indexing of documents or objects is done in order to be searchable
    - * there are many ways to do indexing
  + to index one needs an indexing language
    - * there are many indexing languages
      * even taking every word in a document is an indexing language
* Knowing searching is knowing indexing

**Indexing Subsystem**

Documents

Tokenize

Stop list

Stemming & Normalize

Term weighting

text

non-stop list tokens

tokens

stemmed terms

terms with weights

Assign document identifier

documents

document IDs

**Searching Subsystem**

Index

query

parse query

Stemming & Normalize

stemmed terms

Stop list

non-stoplist tokens

query tokens

Similarity Measure

ranking

Index terms

ranked document set

relevant document set

Term weighting

Query terms

# Chapter Two

## Text/Document Operations & Automatic Indexing

**Index term selection**

* **Indexing** is used to speed up access to desired information from document collection as per users query such that:
  1. It enhances efficiency in terms of time for retrieval. Relevant documents are searched and retrieved quick
* Noun words (or group of noun words) are more representative of the semantics of a doc content
* Preprocess the text of docs in collection in order to select the meaningful/representative index terms
* Control the size of the vocabulary
* *Index* language is the language used to describe documents and requests
* Elements of the index language are *index terms* which may be *derived* from the text of the document to be described, or may be arrived at independently.
  1. If a full text representation of the text is adopted, then all words in the text are used as index terms = full text indexing
  2. Otherwise, need to select the words to be used as index terms for reducing the size of the index file which is basic to design an efficient searching IR system

**Word distribution: Zipf’s Law**

* **Zipf's Law**- named after the Harvard linguistic professor George Kingsley Zipf (1902-1950),
  1. attempts to capture the distribution of the frequencies (i.e. , number of occurances ) of the words within a text.
* **Zipf's Law** states that when the distinct words in a text are arranged in decreasing order of their frequency of occuerence (most frequent words first), the occurence characterstics of the vocabulary can be characterized by the constant rank-frequency law of Zipf:

***Frequency \* Rank = constant***

that is If the words, *w*, in a collection are ranked, *r*, by their frequency, *f*, they roughly fit the relation: **r \* f = c**

* + Different collections have different constants *c*.

***Zipf’s distributions* Rank Frequency Distribution**

For all the words in a collection of documents, for each word *w*

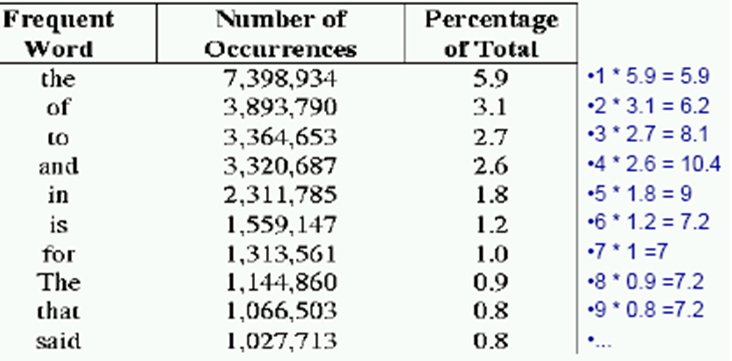
* *f:* is the frequency that *w* appears
* *r:* is rank of *w* in order of frequency. (The most commonly occurring word has rank 1, etc.)

*r*

*w* has rank *r* and frequency *f*

Distribution of sorted word frequencies, according to Zipf’s law

**Example: Zipf's Law**



* The table shows the most frequently occurring words from 336,310 document collection containing 125, 720, 891 total words; out of which 508, 209 unique words

**More Example: Zipf’s Law**

* Illustration of Rank-Frequency Law. Let the total number of word occurrences in the sample N = 1, 000, 000

|  |  |  |  |
| --- | --- | --- | --- |
| **Rank (R)** | **Term** | **Frequency (F)** | **R.(F/N)** |
| 1 | the | 69 971 | 0.070 |
| 2 | of | 36 411 | 0.073 |
| 3 | and | 28 852 | 0.086 |
| 4 | to | 26 149 | 0.104 |
| 5 | a | 23237 | 0.116 |
| 6 | in | 21341 | 0.128 |
| 7 | that | 10595 | 0.074 |
| 8 | is | 10099 | 0.081 |
| 9 | was | 9816 | 0.088 |
| 10 | he | 9543 | 0.095 |

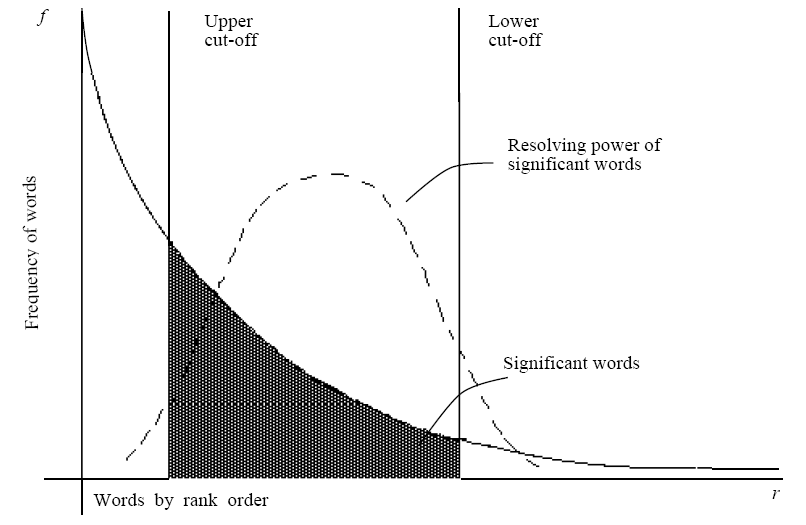
**Methods that Build on Zipf's Law**

* **Stop lists:** Ignore the most frequent words (upper cut-off). *Used by almost all systems.*
* **Significant words:** Take words in between the most frequent (upper cut-off) and least frequent words (lower cut-off).
* **Term weighting:** Give differing weights to terms based on their frequency, with most frequent words weighted less. *Used by almost all ranking methods.*

**Word significance: Luhn’s Ideas**

* Luhn Idea (1958): the frequency of word occurrence in a text furnishes a useful measurement of word significance.
* Luhn suggested that both extremely common and extremely uncommon words were not very useful for indexing.
* For this, Luhn specifies two cut-off points: an upper and a lower cutoff based on which non-significant words are excluded
  1. The words exceeding the upper cut-off were considered to be common
  2. The words below the lower cut-off were considered to be rare
  3. Hence, they are not contributing significantly to the content of the text
  4. The ability of words to discriminate content, reached a peak at a rank order position half way between the two-cutoffs
* Let f be the frequency of occurrence of words in a text, and r their rank in decreasing order of word frequency, then a plot relating f & r yields the following curve

**Luhn’s Ideas**



Luhn (1958) suggested that both extremely common and extremely uncommon words were not very useful for document representation & indexing.

**Text Operations**

* Not all words in a document are equally significant to represent the contents/meanings of a document
  1. Some word carries more meaning than others
  2. Noun words are the most representative of a document content
* Therefore, one needs to preprocess the text of a document in a collection to be used as index terms
* Using the set of all words in a collection to index documents creates too much noise for the retrieval task
  1. Reduce noise means reduce words which can be used to refer to the document
* Preprocessing is the process of controlling the size of the vocabulary or the number of distinct words used as index terms
  1. Preprocessing will lead to an improvement in the information retrieval performance
* However, some search engines on the Web omit preprocessing
  1. Every word in the document is an index term
* Text operations is the process of text transformations in to logical representations
* The main operations for selecting index terms, i.e. to choose words/stems (or groups of words) to be used as indexing terms are:
  1. Lexical analysis/Tokenization of the text - digits, hyphens, punctuations marks, and the case of letters
  2. Elimination of stop words - filter out words which are not useful in the retrieval process
  3. Stemming words - remove affixes (prefixes and suffixes)
  4. Construction of term categorization structures such as thesaurus, to capture relationship for allowing the expansion of the original query with related terms

**Generating Document Representatives**

* Text Processing System
  1. Input text – full text, abstract or title
  2. Output – a document representative adequate for use in an automatic retrieval system
* The document representative consists of a list of class names, each name representing a class of words occurring in the *total* input text. A document will be indexed by a name if one of its *significant* words occurs as a member of that class.

Tokenization

stemming

Thesaurus

Index terms

stop words

documents

**Lexical Analysis/Tokenization of Text**

* Change text of the documents into words to be adopted as index terms
* Objective - identify words in the text
  1. Digits, hyphens, punctuation marks, case of letters
  2. Numbers are not good index terms (like 1910, 1999); but 510 B.C. – unique
  3. Hyphen – break up the words (e.g. state-of-the-art = state of the art)- but some words, e.g. gilt-edged, B-49 - unique words which require hyphens
  4. Punctuation marks – remove totally unless significant, e.g. program code: x.exe and xexe
  5. Case of letters – not important and can convert all to upper or lower

**Tokenization**

* Analyze text into a sequence of discrete tokens (words).
* Input: “*Friends, Romans and Countrymen*”
* Output: Tokens (an instance of a sequence of characters that are grouped together as a useful semantic unit for processing)
  1. *Friends*
  2. *Romans*
  3. *and*
  4. *Countrymen*
* Each such token is now a candidate for an index entry, after further processing
* But what are valid tokens to omit?

**Issues in Tokenization**

* One word or multiple: How do you decide it is one token or two or more?
  1. ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
     1. ***state-of-the-art***: break up hyphenated sequence.
     2. ***San Francisco,*** Los Angeles
     3. Addis Ababa, Arba Minch
  2. ***lowercase***, ***lower-case***, ***lower case***?
     1. data base, database, data-base
  3. Numbers:
     1. dates (3/12/91 vs. Mar. 12, 1991);
     2. phone numbers,
     3. IP addresses (100.2.86.144)
* How to handle special cases involving apostrophes, hyphens etc? C++, C#, URLs, emails, …
  1. Sometimes punctuation (e-mail), numbers (1999), and case (Republican vs. republican) can be a meaningful part of a token.
  2. However, frequently they are not.
* Simplest approach is to ignore all numbers and punctuation and use only case-insensitive unbroken strings of alphabetic characters as tokens.
  1. Generally, don’t index numbers as text, But often very useful. Will often index “meta-data”, including creation date, format, etc. separately
* Issues of tokenization are language specific
  1. Requires the language to be known

**Elimination of STOPWORD**

* Stop-words are extremely common words across document collections that have no discriminatory power
  1. They may occur in 80% of the documents in a collection.
  2. They would appear to be of little value in helping select documents matching a user need and needs to be filtered out from potential index terms
* Examples of stop words are articles, pronouns, prepositions, conjunctions, etc.:
  1. articles (a, an, the); pronouns: (I, he, she, it, their, his)
  2. Some prepositions (on, of, in, about, besides, against, over),
  3. conjunctions/ connectors (and, but, for, nor, or, so, yet),
  4. verbs (is, are, was, were),
  5. adverbs (here, there, out, because, soon, after) and
  6. adjectives (all, any, each, every, few, many, some) can also be treated as stop words
* Stop words are language dependent.

**Stop-words**

* Intuition:
  1. Stop words have little semantic content; It is typical to remove such high-frequency words
  2. Stopwords take up 50% of the text. Hence, document size reduces by 30-50%
* Smaller indices for information retrieval
  1. Good compression techniques for indices: The 30 most common words account for 30% of the tokens in written text
* Better approximation of importance for classification, summarization, etc.

**How to determine a list of stop-words?**

* **One method**: Sort terms (in decreasing order) by collection frequency and take the most frequent ones
  1. Problem: In a collection about insurance practices, “insurance” would be a stop word
* **Another method**: Build a stop word list that contains a set of articles, pronouns, etc.
  1. Why do we need stop lists: With a stop list, we can compare and exclude from index terms entirely the commonest words?
* **With the removal of stop words**, we can measure better approximation of importance for classification, summarization, etc.

**Normalization**

* + It is Canonicalizing tokens so that matches occur despite **superficial differences** in the character sequences of the tokens. It is in a way standardization of text
  + Need to “normalize” terms in indexed text as well as query terms into the same form
  + **Example**: We want to match ***U.S.A.*** and ***USA,*** by deleting periods in a term
* **Case Folding**: Often best to lowercase everything, since users will use lowercase regardless of ‘correct’ capitalization…
  + **Republican** vs. **republican**
  + ***Fasil*** vs. ***fasil vs. FASIL***
  + **Anti-discriminatory** vs. **antidiscriminatory**
  + Car vs. automobile?

**Normalization issues**

* Good for
  1. Allow instances of ***Automobile*** at the beginning of a sentence to match with a query of ***automobile***
  2. Helps a search engine when most users type ***ferrari*** when they are interested in a ***Ferrari*** car
* Bad for
  1. Proper names vs. common nouns
     1. E.g. General Motors, Associated Press, …
* Solution:
  1. lowercase only words at the beginning of the sentence
* In IR, lowercasing is most practical because of the way users issue their queries

**Stemming/Morphological analysis**

* Stemming reduces tokens to their “root” form of words to recognize morphological variation.
  1. The process involves removal of affixes (i.e. prefixes and suffixes) with the aim of reducing variants to the same stem
  2. Often removes inflectional and derivational morphology of a word
     1. Inflectional morphology: vary the form of words in order to express grammatical features, such as singular/plural or past/present tense. *E.g. Boy →* *boys,* *cut → cutting*.
     2. Derivational *morphology*:makes new words from old ones. E.g. c*reation* is formed from *create*, but they are two separate words. And also, *destruction → destroy*
* Stemming is language dependent
  1. Correct stemming is language specific and can be complex.

for example, compressed and

compression is both accepted.

for example, compress and

compress is both accept

**Stemming**

* The final output from a conflation (reducing words to the same token) algorithm is a set of classes, one for each stem detected.
  1. A Stem: the portion of a word which is left after the removal of its affixes (i.e., prefixes and/or suffixes).
  2. Example: ‘connect’ is the stem for {connected, connecting connection, connections}
  3. *Thus, [automate, automatic, automation]*🡪all reduce to 🡪 *automat*
* A class name is assigned to a document if and only if one of its members occurs as a significant word in the text of the document.
  1. A document representative then becomes a list of class names, which are often referred as the documents *index terms/keywords*.
* Queries: Queries are handled in the same way.

**Ways to implement stemming**

There are basically two ways to implement stemming.

* + The first approach is to create a big dictionary that maps words to their stems.
    1. The advantage of this approach is that it works perfectly (insofar as the stem of a word can be defined perfectly); the disadvantages are the space required by the dictionary and the investment required to maintain the dictionary as new words appear.
  + The second approach is to use a set of rules that extract stems from words.
    1. The advantages of this approach are that the code is typically small, and it can gracefully handle new words; the disadvantage is that it occasionally makes mistakes.
    2. But, since stemming is imperfectly defined, anyway, occasional mistakes are tolerable, and the rule-based approach is the one that is generally chosen.

**Porter Stemmer**

* Stemming is the operation of stripping the suffices from a word, leaving its stem.
  1. Google, for instance, uses stemming to search for web pages containing the words ***connected***, ***connecting***, ***connection*** and ***connections*** when users ask for a web page that contains the word ***connect*.**
* In 1979, Martin Porter developed a stemming algorithm that uses a set of rules to extract stems from words, and though it makes some mistakes, most common words seem to work out right.
* Porter describes his algorithm and provides a reference implementation in C at
* It is the most common algorithm for stemming English words to their common grammatical root
* It uses a simple procedure for removing known affixes in English without using a dictionary. To *gets rid of plurals the following* rules are used:
  1. SSES 🡪 SS caresses 🡪 caress
  2. IES 🡪 i ponies 🡪 poni
  3. SS 🡪 SS caress → caress
  4. S 🡪 Φ (nil) cats 🡪 cat
  5. EMENT 🡪 Φ (Delete final *element if what remains is longer* than 1 character)

replacement 🡪 replac

cement 🡪 cement

* *While step 1a gets rid of plurals, step 1b removes -ed or -ing.*
  1. *e.g.*
  2. *; agreed -> agree ; disabled -> disable*

*; matting -> mat ; mating -> mate*

*; meeting -> meet ; milling -> mill  
 ; messing -> mess ; meetings -> mee*

*; feed -> feedt*

**Term weighting**

**Terms**

* Terms are usually stemming. Terms can be also phrases, such as “Computer Science”, “World Wide Web”, etc.
* Documents and queries are represented as vectors or “bags of words” (BOW).
  1. Each vector holds a place for every term in the collection.
  2. Position 1 corresponds to term 1, position 2 to term 2, position n to term n.



W=0 if a term is absent

* Documents are represented by **binary weights** or **Non-binary weighted** vectors of terms.

**Document Collection**

* A collection of *n* documents can be represented in the vector space model by a term-document matrix.
* An entry in the matrix corresponds to the “weight” of a term in the document; zero means the term has no significance in the document or it simply doesn’t exist in the document.

*T1 T2 …. Tt*

*D1 w11 w21 … wt1*

*D2  w12 w22 … wt2*

: : : :

: : : :

*Dn w1n w2n … wtn*

**Binary Weights**

Only the presence (1) or absence (0) of a term is included in the vector

* Binary formula gives every word that appears in a document equal relevance.
  1. It can be useful when frequency is not important.

**Binary Weights Formula:**

**Why use term weighting?**

* Binary weights are too limiting.
  + terms are either present or absent.
  + Not allow to order documents according to their level of relevance for a given query
* Non-binary weights allow to model partial matching.
  + Partial matching allows retrieval of docs that approximate the query.
  + Term-weighting improves quality of answer set.
  + Term weighting enables ranking of retrieved documents; such that best matching documents are ordered at the top as they are more relevant than others.

**Term Weighting: Term Frequency (TF)**

* TF (term frequency) - Count the number of times term occurs in document. *fij* = frequency of term *i* in document *j*
* The more times a term **t** occurs in document **d** the more likely it is that **t** is relevant to the document, i.e. more indicative of the topic.
  + If used alone, it favors common words and long documents.
  + It gives too much credit to words that appears more frequently.
* May want to normalize *term frequency* (*tf*) across the entire corpus:

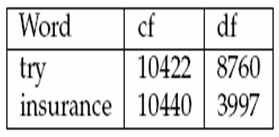
*tfij = fij  / max*{*fij*}

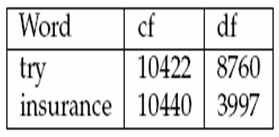


**Document Normalization**

* **Long documents have an unfair advantage:**
  + They use a lot of terms
    - So, they get more matches than short documents
  + And they use the same words repeatedly
    - So, they have much higher term frequencies
* **Normalization seeks to remove these effects:**
  + Related somehow to maximum term frequency.
  + But also sensitive to the number of terms.
* **If we don’t normalize short documents may not be recognized as relevant.**

**Problems with term frequency**

* Need a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance/meaning determination
* Scale down the term weight of terms with high collection frequency
  + Reduce the tf weight of a term by a factor that grows with the collection frequency
* More common for this purpose is document frequency
  + how many documents in the collection contain the term?
* The example shows that collection frequency and document frequency behave differently

**Document Frequency**

* It is defined to be the number of documents in the collection that contain a term

**DF =document frequency**

* + Count the frequency considering the whole collection of documents.
  + Less frequently a term appears in the whole collection, the more discriminating it is.

*df i* = document frequency of term *i*

= number of documents containing term *i*

**Inverse Document Frequency (IDF)**

* *IDF m*easures rarity of the term in collection. The IDF is a measure of the general importance of the term
  + Inverts the document frequency.
* It diminishes the weight of terms that occur very frequently in the collection and increases the weight of terms that occur rarely.
  + Gives full weight to terms that occur in one document only.
  + Gives lowest weight to terms that occur in all documents.
  + Terms that appear in many *different* documents are *less* indicative of overall topic.

*idfi* = inverse document frequency of term *i,*

= log2 (*N/ df i*) (N: total number of documents)

**Inverse Document Frequency**

* E.g.: given a collection of 1000 documents and document frequency, compute IDF for each word?

|  |  |  |  |
| --- | --- | --- | --- |
| **Word** | **N** | **DF** | **IDF** |
| the | 1000 | 1000 | 0 |
| some | 1000 | 100 | 3.322 |
| car | 1000 | 10 | 6.644 |
| merge | 1000 | 1 | 9.966 |

IDF provides high values for rare words and low values for common words.

IDF is an indication of a term’s *discrimination* power.

* + Log used to dampen the effect relative to *tf*.
  + Make the difference between Document frequency vs. corpus frequency?

**TF\*IDF Weighting**

* The most used term-weighting is *tf\*idf weighting scheme*:

*wij = tfij idfi = tfij \** log2 (*N/ dfi*)

* + - A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
  + The tf\*idf value for a term will always be greater than or equal to zero.
    - Experimentally, *tf\*idf* has been found to work well.
  + It is often used in the vector space model together with cosine similarity to determine the similarity between two documents.
* When does TF\*IDF registers a high weight? when a term **t** occurs many times within a small number of documents
  + Highest tf\*idf for a term shows a term has a high term frequency (in the given document) and a low document frequency (in the whole collection of documents); the weights hence tend to filter out common terms.
  + Thus, lending high discriminating power to those documents
* Lower TF\*IDF is registered when the term occurs fewer times in a document, or occurs in many documents
  + Thus, offering a less pronounced relevance signal
* Lowest TF\*IDF is registered when the term occurs in virtually all documents

**Computing TF\*IDF: An Example**

* Assume collection contains 10,000 documents and statistical analysis shows that document frequencies (DF) of three terms are: A (50), B (1300), C (250). And also, term frequencies (TF) of these terms are: A (3), B (2), C (1) with a maximum term frequency of 3. Compute TF\*IDF for each term?

A: tf = 3/3=1.0 idf = log2(10000/50) = 7.644; tf\*idf = 7.644

B: tf = 2/3=0.667 idf = log2(10000/1300) = 2.943; tf\*idf = 1.962

C: tf = 1/3=0.33 idf = log2(10000/250) = 5.322; tf\*idf = 1.774

* Query vector is typically treated as a document and also tf\*idf weighted.

**More Example**

* Consider a document containing 100 words where in the word *cow* appears 3 times. Now, assume we have 10 million documents and *cow* appears in one thousands of these.
  + The term frequency (TF) for *cow*:

3/100 = 0.03

* + The inverse document frequency is

log2(10,000,000 / 1,000) = 13.228

* + The TF\*IDF score is the product of these frequencies: 0.03 \* 13.228 = 0.39684

**Concluding remarks**

* Suppose from a set of English documents, we wish to determine which once are the most relevant to the query "the brown cow."
* A simple way to start out is by eliminating documents that do not contain all three words "the," "brown," and "cow," but this still leaves many documents.
* To further distinguish them, we might count the number of times each term occurs in each document and sum them all together;
  + the number of times a term occurs in a document is called its TF. However, because the term "the" is so common, this will tend to incorrectly emphasize documents which happen to use the word "the" more, without giving enough weight to the more meaningful terms "brown" and "cow".
  + Also, the term "the" is not a good keyword to distinguish relevant and non-relevant documents and terms like "brown" and "cow" that occur rarely are good keywords to distinguish relevant documents from the non-relevant once.

# Chapter Three

## Indexing Structure

**Indexing: Basic concepts**

* Indexing is an arrangement of index terms to permit fast searching and reducing memory space requirement.
* Used to speed up access to desired information from document collection as per user’s query such that:
  + It enhances efficiency in terms of time for retrieval. Relevant documents are searched and retrieved quick.
  + Index file usually has index terms in a sorted order. Which list is easier to search?

***Fox pig zebra hen ant cat dog lion ox***

***ant cat dog fox hen lion ox pig zebra***

* An **index file** consists of records, called **index entries**.
* **Index files** are much smaller than the original file.
  + Remember **Heaps Law**: For 1 GB of text collection the vocabulary has a size of only 5 MB
  + This size may be further reduced by **Linguistic pre-processing** (like stemming & other normalization methods).
    - * The usual unit for indexing is the word
  + **Index terms** - are used to look up records in a file.

**Major Steps in Index Construction**

* **Source file**: Collection of text document
  + A document can be described by a set of representative keywords called index terms.
* **Index Terms Selection**:
  + **Tokenize**: identify words in a document, so that each document is represented by a list of keywords or attributes
  + **Stop words**: removal of high frequency words
    - Stop list of words is used for comparing the input text
  + **Word stem**: reduce words with similar meaning into their stem/root word
  + **Term relevance weight**: Different index terms have varying relevance when used to describe document contents.
    - This effect is captured through the **assignment of numerical weights to each index term** of a document.
    - There are different index terms weighting methods: **TF, TF\*IDF**, …
      * **Output**: a set of index terms (vocabulary) to be used for **Indexing** the documents that each term occurs in.

**Basic Indexing Process**

Token stream.

Modified tokens.

Index File (Inverted file).

***friend***

***roman***

***countryman***

2

4

2

13

16

1

roman

friend

Romans

Friends

Friends, Romans, countrymen.

Indexer

**Linguistic preprocessing**

Tokenizer

Documents to

be indexed.

countrymen

countryman

**Building Index file**

* An index file of a document is a file consisting of a list of index terms and a link to one or more documents that has the index term
* A good **index file** maps each keyword *Ki* to a set of documents D*i* that contain the keyword



* Index file usually has index terms in a sorted order.
  1. The sort order of the terms in the index file provides an order on a physical file
     + - 1. An index file is list of search terms that are organized for associative look-up, i.e., to answer user’s query:
  2. In which documents does a specified search term appear?
  3. Where within each document does each term appear? (There may be several occurrences.)

For organizing index file for a collection of documents, there are various options available:

* 1. Decide what data structure and/or file structure to use. Is its *sequential file*, *inverted file*, *suffix tree*, etc.?

**Index file Evaluation Metrics**

* Running time of the main operation Access/search time
  + How much is the running time to find the required search key from the list?

Update time (Insertion time, Deletion time, ….)

* + How much time does it take to update existing records in an attempt to add a new term or delete existing unnecessary terms?
  + Does the indexing structure allow incremental update or re-indexing?

Space overhead

* + Computer storage space consumed for keeping the list

**Sequential File**

Sequential file is the most primitive file structures. It has no vocabulary as well as linking pointers. The records are generally arranged serially, one after another, but in lexicographic order on the value of some key field. a particular attribute is chosen as primary key whose value will determine the order of the records. when the first key fails to discriminate among records, a second key is chosen to give an order.

**Example:**

* Given a collection of documents, they are parsed to extract words and these are saved with the Document ID.

Doc 1

I did enact Julius

Caesar I was killed

i' the Capitol;

Brutus killed me.

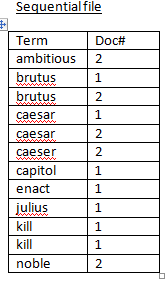
Doc 2

So, let it be with

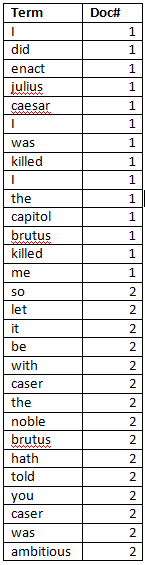
Caesar. The noble

Brutus hath told you

Caesar was ambitious

**Sorting the Vocabulary**

* After all documents have been tokenized, stop words removed and normalization and stemming are applied to generate index terms.
* These index terms in sequential file are sorted in alphabetical order.



**Sequential File**

* To access records, search serially;
  + starting at the first record read and investigate all the succeeding records until the required record is found or end of the file is reached.
  + Its main advantages are:
  + easy to implement;
  + provides fast access to the next record using lexicographic order.
  + Can be searched quickly, e.g., by binary search,
* *Its disadvantages:* 
  + No weights attached to the term.
  + random access is extremely slow: since similar terms are indexed individually, we need to find all terms that match with the query.

**Inverted file**

* A word-oriented indexing mechanism based on sorted list of keywords, with each keyword having links to the documents containing it
  + Building and maintaining an inverted index is a relatively low-cost risk. On a text of n words an inverted index can be built in O(n) time.
* Data to be held in the inverted file includes:
* The vocabulary (List of terms): is the set of all distinct words (index terms) in the text collection.
  + Having information about vocabulary (list of terms) speeds searching for relevant documents.

For each term the inverted file contains information related to:

* + **Location**: all the text location/position where the occurs
  + **Frequency**: occurrence of terms in a document collection
    - *TFij,* number of occurrences of term *tj* in document **d***i*
    - *DFj,* number of documents containing *tj*
    - *mi,* maximum frequency of any term in **d***i*
    - *n,* total number of documents in a collection
    - *CF*, total frequency of *tj* in the corpus n
    - ….
  + 0943326700
* Having information about vocabulary (list of terms)
  + speeds searching for relevant documents
* Having information about the location of each term within the document helps for:
  + user interface design: highlight location of search term
  + proximity based ranking: *adjacency* and *near* operators (in Boolean searching)
    - * + Having information about frequency is used for:
    - calculating term weighting (like TF, TF\*IDF, …)
    - optimizing query processing
* Documents are organized by the terms/words they contain

This is called an **index file**. Text operations are performed before building the index.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Term*** | ***CF*** | ***Document ID*** | ***TF*** | ***Location*** |
| Term 1 | 3 | 2  19  29 | 1  1  1 | 66  213  45 |
| Term 2 | 4 | 3  19  22 | 1  2  1 | 94  7, 212  56 |
| Term 3 | 1 | 5 | 1 | 43 |
| Term 4 | 3 | 11  34 | 2  1 | 3, 70  40 |

**Construction of Inverted file**

An *inverted index* consists of two files: vocabulary and posting files

* A **vocabulary file (Word list):**
  + *stores* all of the **distinct** terms (keywords) that appear in any of the documents (in lexicographical order) and
  + For each word a pointer to posting file
    - * Records kept for each term *j* in the word list contains the following:
  + term *j*
  + Frequency of a term in a given document (TF)
  + number of documents in which term *j* occurs (*DFj*)
  + Total frequency of term j (CF)
  + pointer to inverted (postings) list for term *j*

**Postings File (Inverted List)**

* *For* each distinct term in the vocabulary, stores a list of *pointers* to the documents that contain that term.
* Each element in an inverted list is called a **posting**, i.e., the occurrence of a term in a document
* Each list consists of one or many individual postings

Advantage of dividing inverted file:

* Keeping a pointer in the vocabulary to the list in the posting file allows:
  + the vocabulary to be kept in memory at search time even for large text collection, and
  + Posting file to be kept on disk for accessing to documents

**Organization of Index File**

**Inverted List**

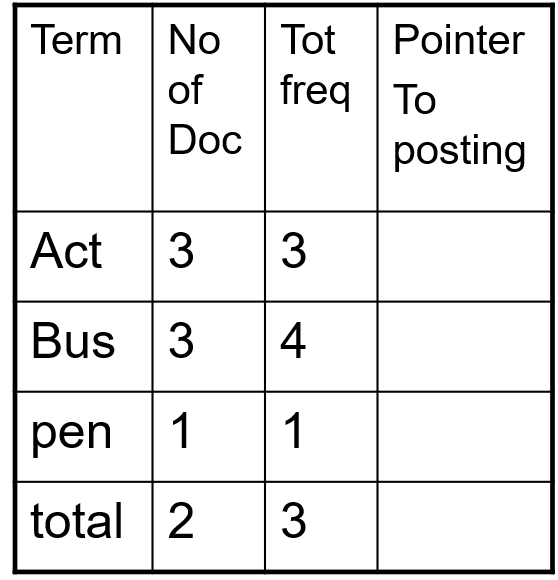
Vocabulary

(word list)

**Postings**

**(inverted list)**

**Documents**



**Example:**

* Given a collection of documents, they are parsed to extract words and these are saved with the Document ID.

**Doc 1**

I did enact Julius

Caesar I was killed

i' the Capitol;

Brutus killed me.

**Doc 2**

So, let it be with

Caesar. The noble

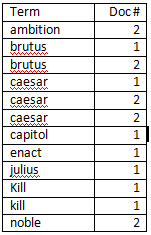
Brutus hath told you

Caesar was ambitious

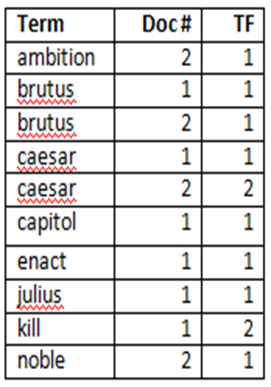


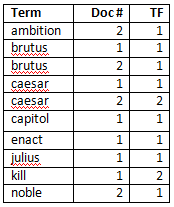
**Sorting the vocabulary**

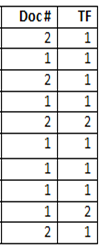
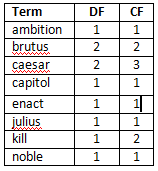
After all document have been tokenized, the inverted file is sorted by terms

**Remove stop words, stemming & compute frequency**

* Multiple term entries in a single document are merged and frequency information added



**Vocabulary & posting file**



**Posting**

**Vocabulary**

Pointers

**Searching on Inverted file**

* Since the whole index file is divided into two, searching can be done faster by loading vocabulary list which takes less memory even for large document collection
* Using binary search, the searching takes logarithmic time
* The search is in vocabulary list
* Updating inverted file is complex
* We need to update both vocabulary and posting file
* What is Suffix? A suffix is a substring that exists at the end of the given string.
  + Each position in the text is considered as a text suffix
  + If txt=t1t2...ti...tn is a string, then Ti=ti, ti+1...tn is the *suffix* of txt that starts at position i,

**Example**: txt = **mississippi**  txt = **GOOGOL**

T1 = mississippi; T1 = GOOGOL

T2 = ississippi; T2 = OOGOL

T3 = ssissippi; T3 = OGOL

T4 = sissippi; T4 = GOL

T5 = issippi; T5 = OL

T6 = ssippi; T6 = L

T7 = sippi;

T8 = ippi;

T9 = ppi;

T10 = pi;

T11 = i;

**Suffix tree**

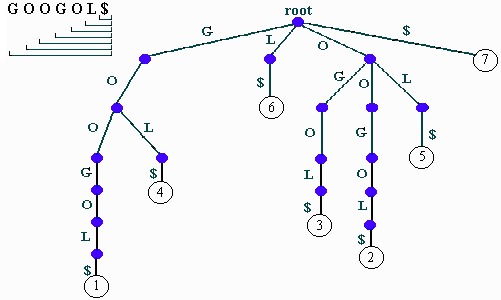
A suffix tree is an ordinary tree in which the input strings are all possible suffixes.

* + **Principles**: The idea behind suffix TRIE is to assign to each symbol in a text an index corresponding to its position in the text. (i.e: First symbol has index 1, last symbol has index n (#of symbols in text).

To build the suffix TRIE we use these indices instead of the actual object.

The structure has several advantages:

* + It requires less storage space.
  + We do not have to worry how the text is represented (binary, ASCII, etc).
  + We do not have to store the same object twice (no duplicate).
* Construct suffix tree for the following string: **GOOGOL**
* We begin by giving a position to every suffix in the text starting from left to right as per characters occurrence in the string.
* TEXT: G O O G O L $  
   POSITION: 1 2 3 4 5 6 7
* Build a SUFFIX TRIE for all n suffixes of the text.
* Note: The resulting tree has n leaves and height n.



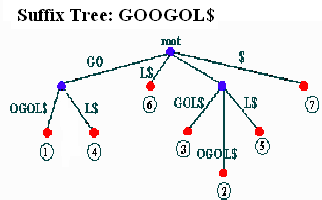
This structure is particularly useful for any application requiring prefix based ("starts with") pattern matching.

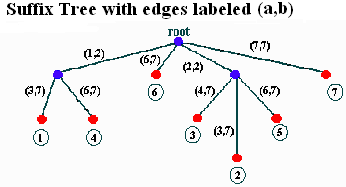
* A suffix tree is a member of the tree family. It is a Tree of all the proper suffixes of S
  + The suffix tree is created by compacting unary nodes of the suffix TRIE.
* We store pointers rather than words in the leaves.
  + It is also possible to replace strings in every edge by a pair (a,b), where a & b are the beginning and end index of the string. i.e.

(3,7) for OGOL$

(1,2) for GO

(7,7) for $





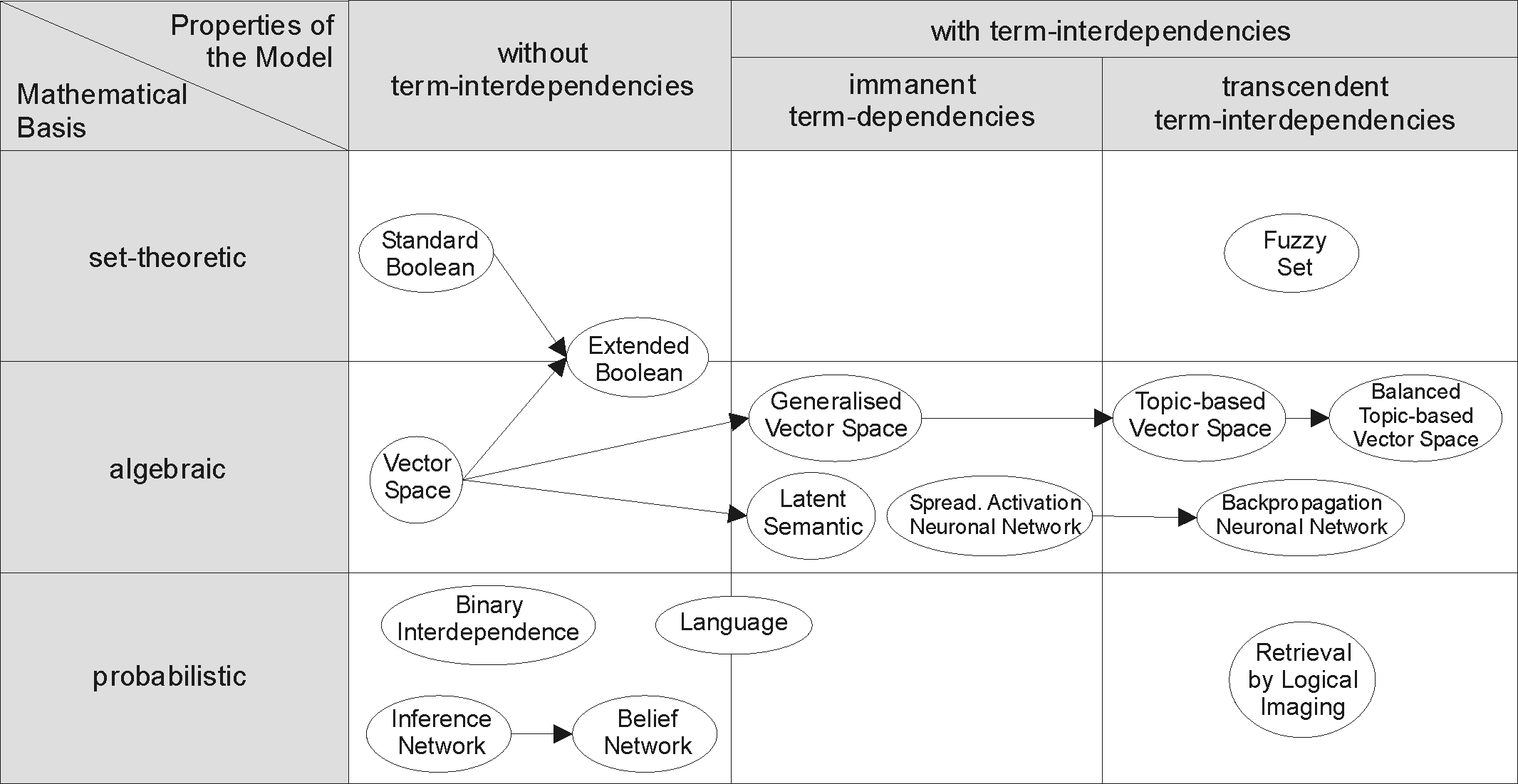
# Chapter four

## IR Models

## IR Models - Basic Concepts

* Word evidence:
  + IR systems usually adopt index terms to index and retrieve documents
  + Each document is represented by a set of representative keywords or index terms (called Bag of Words)
    - * An index term is a document word useful for remembering the document main themes
* Not all terms are equally useful for representing the document contents:
  + less frequent terms allow identifying a narrower set of documents
  + But no ordering information is attached to the Bag of Words identified from the document collection.
* One central problem regarding IR systems is the issue of predicting the degree of relevance of documents for a given query
  + Such a decision is usually dependent on a ranking algorithm which attempts to establish a simple ordering of the documents retrieved
  + Documents appearning at the top of this ordering are considered to be more likely to be relevant
  + Thus ranking algorithms are at the core of IR systems
  + The IR models determine the predictions of what is relevant and what is not, based on the notion of relevance implemented by the system

**Alternative IR models**



* After preprocessing, N distinct terms (Bag of words) remain which are unique terms that form the VOCABULARY
* Let
  + *ki* be an index term i & *dj* be a document j
  + *K = (k1, k2, …, kN)* is the set of all index terms
* Each term, *i*, in a document or query *j*, is given a real-valued weight, *wij.*
  + *wij* is a weight associated with *(ki,dj).* If *wij = 0 ,* it indicates that term does not belong to document *dj*
  + The weight *wij* quantifies the importance of the index term for describing the document contents
  + *vec(dj) = (w1j, w2j, …, wtj)* is a weighted vector associated with the document *dj*

**Mapping Documents & Queries**

* Represent both documents and queries as N-dimensional vectors in a term-document matrix, which shows occurrence of terms in the document collection or query

E.g. 

* An entry in the matrix corresponds to the “weight” of a term in the document; zero means the term doesn’t exist in the document.

*T1 T2 …. TN*

*D1 w11 w12 … w1N*

*D2  w21 w22 … w2N*

*: : : :*

*: : : :*

*DM wM1 wM2 … wMN*

*Qi wi1 wi2 … wiN*

* Document collection is mapped to term-by-document matrix
* View as vector in multidimensional space
  + - Nearby vectors are related
* Normalize for vector length to avoid the effect of document length

**Weighting Terms in Vector Space**

* The *importance* of the index terms is represented by weights associated to them
* **Problem**: to show the importance of the index term for describing the document/query contents, what weight we can assign?
* **Solution** 1: Binary weights: t=1 if presence, 0 otherwise
  + **Similarity**: number of terms in common
* **Problem**: Not all terms equally interesting
  + E.g. the vs. dog vs. cat
* **Solution**: Replace binary weights with non-binary weights



**How to evaluate Models?**

* We need to investigate what procedures they follow and what techniques they used for:
  + Are they using binary or non-binary weighting for measuring importance of terms in documents?
  + Are they using similarity measurements?
  + Are they applying partial matching?
  + Are they performing Exact matching or Best matching for document retrieval?
  + Any Ranking mechanism?

**The Boolean Model**

* Boolean model is a simple model based on set theory
* The Boolean model imposes a binary criterion for deciding relevance
* Terms are either present or absent. Thus, *wij* ε *{0,1}*
* *sim(q,dj) = 1, if document satisfies the boolean query*  *otherwise*

T1 T2 …. TN

D1 w11 w12 … w1N

D2  w21 w22 … w2N

: : : :

: : : :

DM wM1 wM2 … wMN

* *Note that, no weights assigned in-between 0 and 1, just only values 0 or 1*

**d1**

**d2**

**d3**

**d4**

**d5**

**d6**

**d7**

**k1**

**k2**

**k3**

**The Boolean Model: Example**

Generate the relevant documents retrieved by the Boolean model for the query :

*q = k1* ∧ *(k2* ∨ ¬*k3)*

* Given the following determine documents retrieved by the Boolean model based IR system
  + Index Terms: *K*1, *…,K*8*.*
  + Documents:

1. D1 = {K1, K2, K3, K4, K5}

2. D2 = {K1, K2, K3, K4}

3. D3 = {K2, K4, K6, K8}

4. D4 = {K1, K3, K5, K7}

5. D5 = {K4, K5, K6, K7, K8}

6. D6 = {K1, K2, K3, K4}

* + Query: K1Ù (K2 Ú ØK3)
* Answer: {*D*1, *D*2, *D*4, *D*6} Ç ({*D*1, *D*2, *D*3, *D*6} È{*D*3, *D*5})

= {*D*1, *D*2, *D*6}

**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
* The Boolean queries formulated by the users are most often too simplistic
  + As a consequence, the Boolean model frequently returns either too few or too many documents in response to a user query

**Vector-Space Model**

* This is the most commonly used strategy for measuring relevance of documents for a given query. This is because,
  + Use of binary weights is too limiting
  + Non-binary weights provide consideration for partial matches
* These term weights are used to compute a degree of similarity between a query and each document
  + Ranked set of documents provides for better matching
* The idea behind VSM is that
  + the meaning of a document is conveyed by the words used in that document

To find relevant documens for a given query,

* First, map documents and queries into term-document vector space.
  + Note that queries are considered as short document
* Second, in the vector space, queries and documents are represented as weighted vectors, *wij* 
  + There are different weighting technique; the most widely used one is computing tf\*idf for each term
* Third, similarity measurement is used to rank documents by the closeness of their vectors to the query.
  + Documents are ranked by closeness to the query. Closeness is determined by a similarity score calculation

**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 we can control the size of the retrieved set of documents.

**Vector-Space Model**

* Advantages:
  + term-weighting improves quality of the answer set since it displays in ranked order
  + partial matching allows retrieval of documents that approximate the query conditions
  + cosine ranking formula sorts documents according to degree of similarity to the query
* Disadvantages:
  + assumes independence of index terms

**Probabilistic Model**

* IR is an uncertain process
  + Mapping Information need to Query is not perfect
  + Mapping Documents to index terms is a logical representation
  + Query terms and index terms mostly mismatch
* This situation leads to several statistical approaches: probability theory, fuzzy logic, theory of evidence, etc.
* Probabilistic retrieval model is rigorous formal model that attempts to predict the probability that a given document will be relevant to a given query (P(R|q,di)
  + Use probability to estimate the “odds” of relevance of a query to a document.
  + It relies on accurate estimates of probabilities
* Asks the question: what is the probability that user will see relevant information if they read this document.
  + *P(rel | di ):* probability of relevance after reading *di*
  + How likely is the user to get relevance information from reading this document?
  + high probability means more likely to get relevant info.
* A Probabilistic retrieval models
  + Rank documents in decreasing order of probability of relevance to user’s information need
  + Calculate P(*rel|di) for each document and rank*

**Probability Ranking Principle**

* You have a collection of Documents
  + User issues a query
  + A Set of documents needs to be returned
  + Intuitively, want the “best” document to be first, second best - second, etc…
  + We need a formal way to judge the “goodness” of documents with respect to queries.
* Probability ranking principle: if a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance… the overall effectiveness of the system to its user will be the best that is obtainable.

**Difficulties**

* Evidence is based on a lossy representation
  + Evaluate probability of relevance based on occurrence of terms in query and documents
  + Start with an initial estimate, refine through relevance feedback
* Computing the probabilities exactly according to the model is intractable
  + Make some simplifying assumptions

**Probabilistic model**

* Probabilistic model uses probability theory to model the uncertainty in the retrieval process
  + Assumptions are made explicit
  + Term weight without relevance information is IDF
* Relevance feedback can improve the ranking by giving better term probability estimates
* Advantages of probabilistic model over vector‐space
  + Strong theoretical basis
  + Based on probability theory (very well understood)
  + Easy to extend
* Disadvantages
  + Models are often complicated
  + No term frequency weighting
* Which is better: vector‐space or probabilistic?
  + Both are approximately as good as each other
  + Depends on collection, query, and other factors

# Chapter Five

## Retrieval evaluation

***Introduction***

* Evaluation is a systematic determination of a subject’s merit, worth and significance using criteria governed by a set of standards.
* It ascertains the degree of achievement in regard to the aim and objectives and results of any such actions that has been completed.
* Evaluation of IR systems measure which of the two existing system perform better and try to assess how the level of performance of a given can be improved
* Evaluation of IR system can be justified by the following three issues:
  + How well the system is satisfying its objectives
  + How efficiently it is satisfying its objectives
  + Whether the system justified its existence

***Why system evaluation***

* Any systems need validation and verification
  + Check whether the system is right or not
  + Check whether it is the right system or not
* It provides the ability to measure the difference between IR systems
  + How well do our search engines work?
  + Is system A better than B?
  + Under what conditions?
  + Evaluation drives what to research
  + Identify techniques that work and do not work
  + There are many retrieval models/ algorithms/ systems
    1. which one is the best?
  1. What is the best component for?
     1. Similarity measures (dot-product, cosine, …)
     2. Index term selection (stop-word removal, stemming…)
     3. Term weighting (TF, TF-IDF,)

***Evaluation Criteria***

What are some main measures for evaluating an IR system’s performance?

* Efficiency: time and space complexity
  1. Speed in terms of retrieval time and indexing time
  2. Speed of query processing
  3. The space taken by corpus vs. index
     + Is there a need for compression?
  4. Index size: Index/corpus size ratio
     + Effectiveness
  5. How is a system capable of retrieving relevant documents from the collection?
  6. Is system X better than another one?
  7. User satisfaction: How “good” are the documents that are returned as a response to user query?
  8. **Relevance** of results to meet information need of users

***Types of Evaluation Strategies***

* System-centered evaluation
  1. Given documents, queries, and relevance judgments
     1. Try several variations of the system
     2. Measure which system returns the “best” hit list
        1. User-centered evaluation
  2. Given several users, and at least two retrieval systems
     1. Have each user try the same task on both systems
     2. Measure which system works the “best” for user’s information need
     3. How to measure user’s satisfaction?

**The Notion of Relevance Judgment**

* Relevance is a relation between a document and query.
  1. Construct document - query as determined by (i) the user who posed the retrieval problem; (ii) an external judge; (iii) information specialist
  2. Is the relevance judgment made by users and external person the same?
  3. Relevance judgment is usually:
  4. **Subjective**: Depends upon a specific user’s judgment.
  5. **Situational**: Relates to user’s current needs.
  6. **Cognitive**: Depends on human perception and behavior.
  7. **Dynamic**: Changes over time.

**Measuring Retrieval Effectiveness**

* Retrieval of documents may result in:
  1. **False negative (false drop):** some relevant documents may not be retrieved.
  2. **False positive:** some irrelevant documents may be retrieved.
  3. For many applications a good index should not permit any false drops, but may permit a few false positives.

“Type one errors”

“Errors of commission” “False positives”

Relevant

Irrelevant

retrieved

Not

retrieved

**A**

**B**

**C**

**D**

“Type two errors”

“Errors of omission” “False negatives”

* Metrics often used to evaluate effectiveness of the system

“Type one errors”

“Errors of commission” “False positives”

**Precision & Recall**

* **Precision**

The ability to retrievetop-ranked documents that are mostly relevant.

retrieved & relevant

not retrieved but relevant

retrieved & irrelevant

Not retrieved & irrelevant

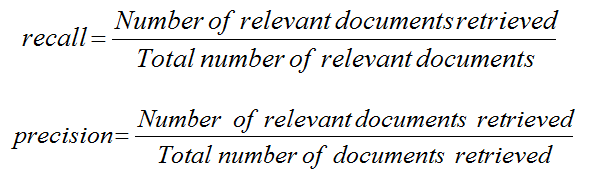
retrieved

not retrieved

relevant

irrelevant

* **Recall**

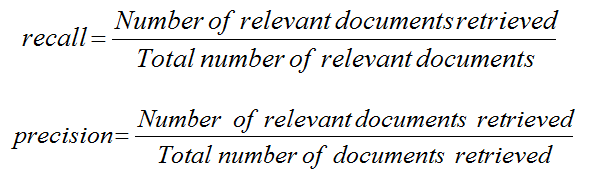
The ability of the search to find ***all*** of the relevant items in the corpus.

**Relevant documents**

**Retrieved documents**

Entire document collection

**Relevant +retrieved**



**Example**

* Assume that there is a total of 10 relevant documents

|  |  |  |  |
| --- | --- | --- | --- |
| **Ranking** | **Retrieval** | **Recall** | **Precision** |
| 1. Doc. 50 | R | 0.10 | 1.00 |
| 2. Doc. 34 | NR | 0.10 | 0.50 |
| 3. Doc. 45 | R | 0.20 | 0.67 |
| 4. Doc. 8 | NR | 0.20 | 0.50 |
| 5. Doc. 23 | NR | 0.20 | 0.40 |
| 6. Doc. 16 | NR | 0.20 | 0.33 |
| 7. Doc. 63 | R | 0.30 | 0.43 |
| 8. Doc. 119 | R | 0.40 | 0.50 |
| 9. Doc. 21 | NR | 0.40 | 0.44 |
| 10. Doc. 80 | R | 0.50 | 0.50 |

**Graphing Precision and Recall**

* Plot each (recall, precision) point on a graph
* Recall is a non-decreasing function of the number of documents retrieved,
* precision usually decreases (in a good system)

**Precision/Recall tradeoff**

* Can increase recall by retrieving many documents (down to a low level of relevance ranking),
  + - but many irrelevant documents would be fetched, reducing precision
* Can get high recall (but low precision) by retrieving all documents for all queries

Recall

Precision

The ideal

Returns relevant documents but

misses many useful ones too

Returns most relevant

documents but includes

lots of junk

***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.

# Chapter six (6)

## Query Languages

Keyword-based querying

* **Queries** are combinations of words.
* The document collection is searched for documents that contain these words.
* Word queries are intuitive, easy to express and provide fast ranking.
  + The concept of *word* must be defined.
  + A word is a sequence of *letters* terminated by a *separator* (period, comma, space, etc).
  + Definition of *letter* and *separator* is flexible; e.g., hyphen could be defined as a letter or as a separator.
  + Usually, common words (such as “a”, “the”, “of”, …) are ignored.

***Single-word queries***

* A query is a single word
  + Usually used for searching in document images
* Simplest form of query.
* All documents that include this word are retrieved.
* Documents may be ranked by the *frequency* of this word in the document.

***Phrase queries***

* A query is a sequence of words treated as a single unit.
  + Also called “literal string” or “exact phrase” query.
* Phrase is usually surrounded by quotation marks.
* All documents that include this phrase are retrieved.
* Usually, separators (commas, colons, etc.) and common words (e.g., “a”, “the”, “of”, “for” …) in the phrase are ignored.
* In effect, this query is for a *set* of words that must appear *in sequence*.
  + Allows users to specify a *context* and thus gain precision.
* *Example*: “Information Processing for Document Retrieval”.

**Multiple-word queries**

* A query is a set of words (or phrases).
* Two options: A document is retrieved if it includes
  + any of the query words, or
  + each of the query words.
* Documents are ranked by the number of query words they contain:
  + A document containing n query words is ranked higher than a document containing m < n query words.
  + Documents are ranked in decreasing order:
    - those containing all the query words are ranked at the top, only one query word at bottom.
  + Frequency counts may be used to break tie among documents that contain the same query words.
  + Example: what is the result for the query “Red Bird”?

**Boolean queries**

* Based on concepts from logic: AND, OR, NOT
  + *It d*escribes the information needed by relating multiple words with Boolean operators.
* **Semantics**: For each query word *w* a corresponding set *Dw* is constructed that includes the documents that contain *w*.
* The Boolean expression is then interpreted as an expression on the corresponding document sets with corresponding set operators:
  + A**ND:** Finds only documents containing all of the specified words or phrases.
  + O**R:** Finds documents containing at least one of the specified words or phrases.
  + **NOT:** Excludes documents containing the specified word or phrase.

***Examples:* Boolean queries**

*1.computer* ***OR*** *server*

* + *Finds documents containing either computer, server or both*

*2. (computer* ***OR*** *server)* ***NOT*** *mainframe*

* + *Select all documents that discuss computers or servers, do not select any documents that discuss mainframes.*

*3. computer* ***NOT*** *(server* ***OR*** *mainframe)*

* + *Select all documents that discuss computers, and do not discuss either servers or mainframes.*

*4. computer* ***OR*** *server* ***NOT*** *mainframe*

* + *Select all documents that discuss computers, or documents that discuss servers but do not discuss mainframes.*

**Weighted queries**

* Each of the words is assigned a different *weight*, expressing the relative importance of the word within the query.
* A query is then a set of word-weight pairs:

(*k*1, *w*1), …, (*k*n, *w*n).

* The ranking of a document is the *sum* of the weights for the query words that it satisfies.
  + - ***Example***: given Query: (*A*,0.8,), (*B*,0.9), (*C*,0.3); and

Document 1: (*A*, *B*, *D*) and Document 2: (*A*, *C*, *D*) which document ranked first ?

* + Rank of Document 1: 0.8+0.9 = 1.7
  + Rank of Document 2: 0.8+0.3 = 1.1
  + Each document includes two words from the query, but Document1 is ranked higher because it includes more important words.

**Penalizing documents**

* When interpreting queries, some models *demote* documents that include keywords that were not requested
  + The vector model with the cosine measure
  + The Bayesian network model
* *Example*: Assume the vector model with the cosine measure and the simple case that both documents and queries use binary values. Consider these two documents and a query:
  + *d*1 = (0,1,0,1,0), *d*2= (0,1,1,1,0), *q*= (0,1,0,1,0)
  + *sim*(*q*, *d*1) = 1.0, *sim*(*q*, *d*2) = 0.82
  + *d*2 is demoted because it includes an extra keyword not requested by *q*.
* In contrast, the Boolean model does not “penalize” documents with extra (non-requested) keywords
* When interpreting queries, some models *demote* documents that include keywords that were not requested
  + The vector model with the cosine measure
  + The Bayesian network model
* *Example*: Assume the vector model with the cosine measure and the simple case that both documents and queries use binary values. Consider these two documents and a query:
  + *d*1 = (0,1,0,1,0), *d*2= (0,1,1,1,0), *q*= (0,1,0,1,0)
  + *sim*(*q*, *d*1) = 1.0, *sim*(*q*, *d*2) = 0.82
  + *d*2 is demoted because it includes an extra keyword not requested by *q*.
* In contrast, the Boolean model does not “penalize” documents with extra (non-requested) keywords

**Pattern queries**

* ***What is Pattern?***An expression that defines a *set* of words
* ***Pattern matching***: A word *matches* a pattern if it is equal to *one* of the words defined by the pattern.
  + In other words, the semantics are of *disjunction*: A pattern *P* that defines *w*1, *w*2, …, *w*nis interpreted as *w*1 v *w*2 v … v *w*n.
* ***Similarity pattern*** : Specifies a string and a *radius* 
  + *defines* all the words whose *distance from* the string is within the radius.
  + Assume the distance between two strings is measured by the number of one-character changes (insertions, deletions, replacements or transpositions) required to transform one string into the other.
    - The similarity pattern (*king*, 2) defines *kin*, *kong*, *knig*, *kings*, *cling*, …
  + Useful to compensate for typing or scanning (OCR) errors.
  + One of the techniques used for pattern matching is **string editing**

**String editing**

* The problem is given two sequences of symbols, X = x1 x2 …xn andY = y1 y2 …ym, transform X to Y, based on a sequence of three operations: Delete, Insert and Replace, so that for every operation COST(Cij) is incurred.
  + - The objective of string editing is to identify a minimum cost sequence of edit operation that will transform X into Y.

Example: consider the sequences

X = {a a b a b} and Y = {b a b b}

Identify a minimum cost sequence of edit operation that transform X into Y. Assume change costs 2 units, delete 1 unit and insert 1 unit.

**Dynamic programming**

* The minimum cost of any edit sequence that transforms x1 x2 …xi into y1 y2 …yj (for i>0 and j>0) is the minimum of the three costs: delete, change, or insert operations.

Transform the sequences

Xi = {a a b a b} into Yj = {b a b b}

with minimum cost sequence of edit operation using dynamic programming approach, assume that change costs 2 units, delete and insert 1 unit.

The value 3 at (5,4) is the optimal solution

By tracing back one can determine which operations lead to optimal solution

* Delete x1, delete x2 and Insert y4 Or,
* Change x1 to y1 & Delete x4

j

j

4

3

2

1

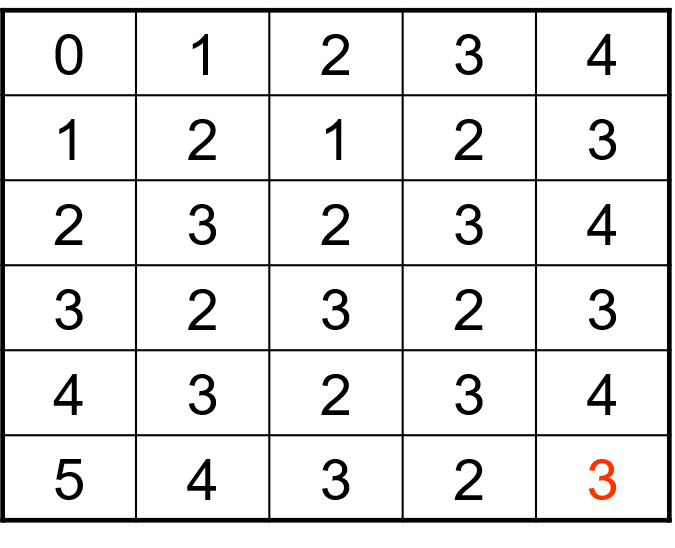
0

The value 3 at (5,4) is the optimal solution

By tracing back one can determine which operations lead to optimal solution

* Delete x1, Delete x2 and Insert y4 Or,
* Change x1 to y1 & Delete x4

4

****

5

4

0

3

2

1

**Natural language**

* Using natural language for querying is very attractive.
* *Example*: Find all the documents that discuss “campaign finance reforms, including documents that discuss violations of campaign financing regulations. Do not include documents that discuss campaign contributions by the gun and the tobacco industries”.
* Natural language queries are converted to a formal language for processing against a set of documents.
* Such translation requires ***intelligence***and is still a challenge
* *Pseudo NL processing*: System scans the text and extracts recognized terms and Boolean connectors. The grammaticality of the text is not important.
  + Often used by search engines.
* *Problem*: Recognizing the negation in the search statement (“Do not include...”).
* *Compromise*: Users enter natural language *clauses* connected with Boolean operators.
* In the above example: “campaign finance reforms” **or** “violations of campaign financing regulations" and **not** “campaign contributions by the gun and the tobacco industries”.

# Chapter Seven (7)

## Query Operations

## Relevance Feedback & Query Expansion

## Introduction

* No detailed knowledge of collection and searching environment
  + - difficult to formulate queries well designed for searching
    - need many formulations of queries for effective searching
    - First formulation: often naïve attempt to retrieve relevant information
* Documents initially retrieved:
  + can be examined for relevance information (by the user or the system automatically) to provide relevance feedback
    - improve query formulations for retrieving additional relevant documents

**Query Reformulation**

* Identify terms related to query terms
* Revise query to account for feedback:
  + Query Expansion: Add new terms related to query terms from relevant documents.
  + Term Reweighting: modify term weights based on documents relevance for the users query
    - Increase weight of terms in relevant documents and decrease weight of terms in irrelevant documents.

Several algorithms for query reformulation

**Approaches for Query Operations**

* Users relevance feedback
  + Approaches based on feedback from users about relevance of documents retrieved
* Pseudo-relevance feedback
  + Approaches based on information derived from set of initially retrieved documents (local set of documents), which is called Local Analysis
  + Approaches based on global information derived from document collection, which is called Global Analysis

**Users Relevance Feedback**

* Most popular query reformulation strategy
* Cycle:
  + User presented with list of retrieved documents
  + User marks those which are relevant
    - In practice: top 10-20 ranked documents are examined
  + Select important terms from documents assessed relevant by users
  + Enhance importance of these terms in a new query
* Expected:
  + New query moves towards relevant documents and away from non-relevant documents

**Relevance Feedback**

* After initial searching results are presented, allow the user to provide feedback on the relevance of one or more of the retrieved documents.
* Use this feedback information to reformulate the query.
* Produce new results based on reformulated query.
* Allows more interactive, multi-pass process.

**Pseudo Relevance Feedback**

* Use relevance feedback methods without explicit user input.
  + Obtain relevance feedback automatically
  + Identify terms related to query terms (e.g. synonyms, stemming variations, terms close to query terms in text)
* Just **assume** the top *m* retrieved documents are relevant, and use them to reformulate the query.
* Allows for query expansion that includes terms that are correlated with the query terms.
* Two strategies
  + Local strategies
  + Global strategies

**Local Analysis**

* Examine only documents retrieved automatically for query to determine query expansion
* At query time, dynamically determine similar terms based on analysis of top-ranked retrieved documents.
* Base correlation analysis on only the “local” set of retrieved documents for a specific query.
* Avoids ambiguity by determining similar (correlated) terms only within relevant documents.
  + “Apple computer” → “Apple computer Powerbook laptop”

**Global analysis**

* Expand query using information from whole set of documents in collection
  + Determine term similarity through a pre-computed statistical analysis of the complete corpus.
* Thesaurus-like structure using all documents
  + Approach to automatically built thesaurus
    - (e.g. similarity thesaurus based on co-occurrence frequency)
  + Approach to select terms for query expansion

**Thesaurus**

* A thesaurus provides information on synonyms and semantically related words and phrases.
* Example:

physician

similar/synonymous: doctor, medical, MD

related: general practitioner, surgeon

* Statistical Thesaurus:
  + Existing human-developed thesauri are not easily available in all languages.
  + Human thesauri are limited in the type and range of synonymy and semantic relations they represent.
  + Semantically related terms can be discovered from statistical analysis of corpora.

**Thesaurus-based Query Expansion**

* For each term, *t*, in a query, expand the query with synonyms and related words of *t* from the thesaurus.
* May weight added terms less than original query terms.
* Generally, increases recall.
* May significantly decrease precision, particularly with ambiguous terms.
  + “interest rate” → “interest rate fascinates evaluate”

**Global vs. Local Analysis**

* Global analysis requires intensive term correlation computation only once at system development time.
  + Local analysis requires intensive term correlation computation for every query at run time (although number of terms and documents is less than in global analysis).
* But local analysis gives better results.
  + Term ambiguity may introduce irrelevant statistically correlated terms during global analysis.
  + “Apple computer” → “Apple red fruit computer”

**Global Analysis Refinements**

* Only expand query with terms that are similar to *all* terms in the query.
  + “fruit” not added to “Apple computer” since it is far from “computer.”
  + “fruit” added to “apple pie” since “fruit” close to both “apple” and “pie.”
* Use more sophisticated term weights (instead of just frequency) when computing term correlations.

**Query Expansion Conclusions**

* Expansion of queries with related terms can improve performance, particularly recall.
* However, must select similar terms very carefully to avoid problems, such as loss of precision.