APPLICATIONS 2:

INFORMATION RETRIEVAL

Theme

The theory and systems developed to filter and retrieve sought-for information from large collections of texts. Large text collections, in text corpora and databases, since late 1950s. Resulting need for quick and accurate methods to find appropriate information. Major aspects:

- text collection, indexing, etc.
- query processing and matching
- text clustering
- multilingual IR
- Web search

Summary of Contents

1. History

From the beginning, librarians of the Information Science field were interested in using computers to index their material. They worked with CS researchers to design clever methods for indexing their documents (only important words, etc.) using typical database indexing and management techniques. Then, in the 1950s, someone named Cleverdon performed the famous Cranfield experiments, in which he showed that the best indexing mechanism simply used *all* the words! Shortly after, in the late 1950s with Gerald Salton at Cornell, Karen Spärck Jones at Cambridge, and others, a new version of processing started, and the field split. The more library and index-oriented side went one way; the more 'statistical' word-level hackers the other, to form SIGIR, devoted to **Information Retrieval** (IR). Currently the latter has strong research arms at Cornell, UMass Amherst, CMU, and elsewhere.

The need for text management in the military and intelligence communities gave rise to research funding and the TREC evaluation competitions, run by Donna Harman and Ellen Voorhees at NIST. With the advent of the internet, text search became something the non-computer person also needed: hence the web crawlers (Yahoo!, AltaVista, Google). Still, performance hovers around 40%. Today, IR is slowly thinking about using semantics. New companies promise enhanced performance using semantics. Others pinpoint results (for example AskJeeves.com).

Straightforward but still impossible dream: parse/analyze texts and queries and then match them. This involves semantics, inevitably. But the IR community is still very suspicious of semantics. So their approximation: word-level processing.

2. Core IR

The three central technical issues: indexing, query handling, and large-scale text management.

2.1. Indexing

The problem is to create an index structure pointing into the set of documents, so that given a query term, you very quickly find all documents containing that (and related) terms. Several issues:

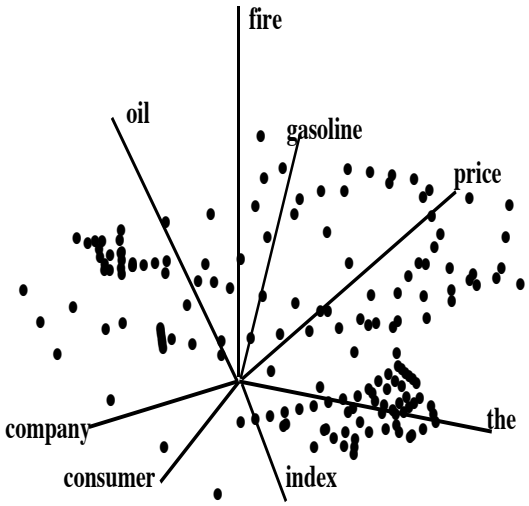
- index structure creation

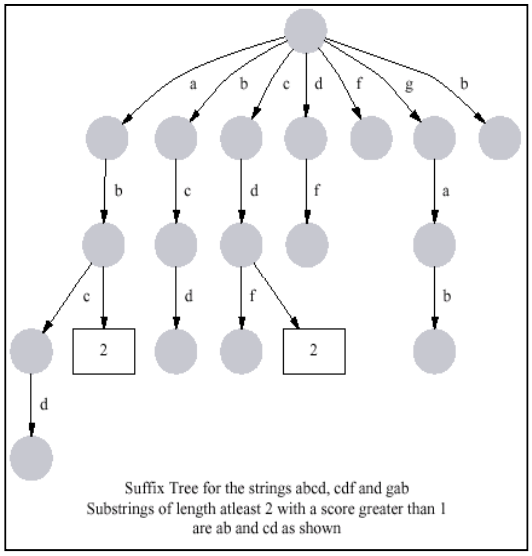
- index structure size
- access speed

Inverted files: Create a list of all the words in the texts, and then for each word create a list of pointers to the positions (doc#; word#) where it occurs. The wordlist memory size is O(n^{β}) where n is the number of words and β is between 0.4 and 0.6. But there must be one pointer for every occurrence of every word in the text!—O(n). And even if you don't index stopwords you still need about 35% of the total collection size. To reduce size you can break up the collection into blocks and then just point to the blocks, letting some other search mechanism take it from there. It's wasteful.

Simple (table) approach: build a table, say a hash table, linking each word with all its occurrences (doc,pos). Hash tables use about 15% of the total collection size.

PAT trees: View the set of texts as a linear string of words (or letters or bits); then each word occurs at some position and is the start of a semi-infinite (sub)string called a *sistring*. Each sistring is unique. Now create a binary tree that indexes the sistrings; as you traverse from the root down, you choose based on the prefix of the string you are interested in. (If you're working at the bit level, the trees are obviously binary.) Leaves of the PAT tree point into document set at right places; subtree's sibling leaves point to all the places that start with the prefix matched so far. Quite expensive to build (O(n) steps, where n is the number of letters), although you can break up the tree into a hierarchically cascaded set of trees. For indexing this is nice to have.





Vector spaces: Define a space of N dimensions, one for each word. Each text is represented by a point at the coordinates of the count of each word, along its dimension. When points are close to each other, their documents have roughly the same frequency distribution of words (and are assumed to be similar).

Represent each document as vector of N words (word frequency, simple or weighted)

D1: <12 37 0 0 0 0 9 0 0 1 0 1 0 0 18 0 0 0 44 …>

Index doc collection in this space

D1, D2, D3…

(Optional) create inverse index

"big" [D1 D12 D19 D37…]

How to measure distance? Need to choose a distance metric. Obvious one is Euclidean distance (using Pythagoras's Theorem). However, there's a problem with the number of terms in a document—short or long documents with the same relative frequencies of words should be in the 'same' place in the space. To overcome this problem, we project the space from the origin onto the unit hypersphere. Then instead of distance between points, you can just measure the angle between the unit vectors that points to them. This is called the Cosine measure (inner product), which is the most common method used. The Cosine distance S_{ij} between document *i* and document *j* is

$$S_{ij} = \frac{\sum_k w_{ik} . w_{jk}}{\sqrt{\sum_k w_{ik}^2 \cdot \sum_k w_{jk}^2}}$$

where *Wik* = freq of word *k* in document *i* . Other similar popular ones are the Dice and Jaccard measures.

To find a document (in answer to the user's query), the query words are plotted in the space, and the system simply takes a hypersphere around that point and return all documents within it (for the Euclidean method), or all documents that are within a small enough cosine angle from it. This simple method is amazingly effective.

Which terms to index on?

Use all words? Only 'important' ones? Typically systems remove 'stop words' such as "the", "an", "he", "and", etc. (What about "not"?)

Still some words are more important than others; you can look at the background frequency distribution of all the words and discover that "man" and "say" are much more frequent than "aardvark" and "hydrochemical". This means that when you see one of the latter in a text, they should be more important.

The simplest and still most common way of working with this is to consider two factors (Sparck Jones):

- **Term frequency** (*tf*) = number of times word *j* appears in the collection
- **Inverse document frequency** (*idf*) = a measure related to the number of different documents that word *j* appears in: $idf_j = log(N/n_j)$

The product of these two, *tf.idf*, rewards words that are unusually frequent. Specifically:

$w_{jk} = tf_j \, . \, idf_j$

tf_{jk} : count of term j in text *k* ("waiter" often only in some texts).

$idf_j = log(N/n_j)$: within-collection frequency ("the" often in <u>all</u> texts),
n_j = number of docs with term *j* , *N* = total number of documents.

tf.idf is the best among 287 methods (Salton & Buckley 88).

Another scoring function is χ^2 ("chi-squared"):

χ^2 : $w_{jk} = (tf_{jk} - m_{jk})^2 / m_{jk}$ if $tf_{jk} > m_{jk}$
0 otherwise

$m_{jk} = (\Sigma_j \, tf_{jk} \, \Sigma_k \, tf_{jk} \,) \; / \; \Sigma_{jk} \, tf_{jk}$: mean count for term *j* in text *k*

Stemming: In addition to counting frequencies, one can consider compressing word variants to their root form. Various stemmers (demorphers) exist for English, some being more aggressive than others (Porter stemmer, for example). Stemmers may give problems though.

Latent Semantic Indexing: Usually, despite all kinds of word reduction steps, you still find that some words co-vary. For an extreme example, you will always find "Dumpty" when you find "Humpty"—the second word adds no new information. Is there a way to compress out the words that are 'predictable' from (combinations of) other words? Yes, using a technique from Signal Processing called singular value decomposition. This is a matrix manipulation technique to reduce the complexity of the document space that in IR is called latent semantic indexing (it uncovers the 'latent' semantics behind co-varying words).

- Create matrix A, one text per column (words are rows).
- Apply SVDPAC to compute matrix U so that $A = U \Sigma U^T$:
    - U : $m \times n$ orthonormal matrix of left singular vectors that span space
    - U^T : $n \times n$ orthonormal matrix of right singular vectors
    - Σ : diagonal matrix with exactly *rank(A)* nonzero singular values; $\sigma_1 > \sigma_2 > \ldots > \sigma_n$
- Take only the first *k* of the new concepts: $\Sigma' = \{\sigma_1, \sigma_2 \ldots \sigma_k\}$.
- Create matrix A′ out of these *k* vectors: $A' = U \Sigma' U^T \approx A$.
- A′ is a new (words × docs) matrix, with different weights and new 'documents'. Each column is a 'purified' document, with only the 'independent' words remaining to be indexed.

Ontologies/topic taxonomies: some of the web search companies use large topic taxonomies and store documents in terms of their terms/leaves; see below.

2.2. Query Handling

The user must specify what is required. For much of IR, the user's input (the *query*) is about a paragraph long (while in web search it is a few words). The problem is to understand what exactly the user wants out of the query that is input. The more terms, the more accurate. Scale of complexity:

- all words, unaltered
- drop stopwords (do you drop "not"?)
- demorph non-stopwords
- allow multi-word units

Boolean queries are one of the oldest forms of query, much used by librarians. Here you combine search terms with AND, OR, and NOT. The problem with Boolean matching is that it is all or nothing—there are no partial results. So it works best with tables and PAT trees, but doesn't use the full potential of vector spaces.

Non-Boolean queries allow you to combine words with varying strengths, say by inserting a whole (short) document and saying "more like this". The frequency distribution of the terms in the query document help specify the relative importance of the search words, mapping directly into the vector space.

Term disambiguation: You can enhance the query by allowing term disambiguation, for example asking whether an ambiguous term is a noun or a verb, or presenting the user with the possible meanings (say, drawn from WordNet). Example: "capital of Switzerland" — do you mean "capital = major city" or "capital = money"? AskJeeves.com uses a kind of frozen dialogue, asking the user to select from a predefined set of categories of meaning for a term/query. When you know which meaning (sense) of the term is intended, you can add appropriate related words—see term expansion.

Term expansion: The idea is to add to the user's query term additional related ones, n order to focus the search. The problem is to find the sets of related words (word families or topic 'signatures'). In general,

this may require a lot of work beforehand, and some storage for each word in the lexicon. Various methods have been proposed to create term expansion lists; see (Lin and Hovy 2000) for example.

Pseudo-relevance feedback: retrieve some documents with user query, select top-scoring *N* of them, and extract best words (using *tf.idf*, etc.). Use these words as query words to perform a *second* retrieval and return results to the user. Works ok; adds to score.

2.3. Presentation of results / interface issues

This is a very important aspect. More research needs to be done.

3. Evaluation

The TREC competitions, run by NIST, started in 1991. Every year, NIST issues a set of queries against a fixed corpus of around 1 million newspaper and other documents, and competing systems return the results, which are graded for Recall and Precision.

What must you measure?

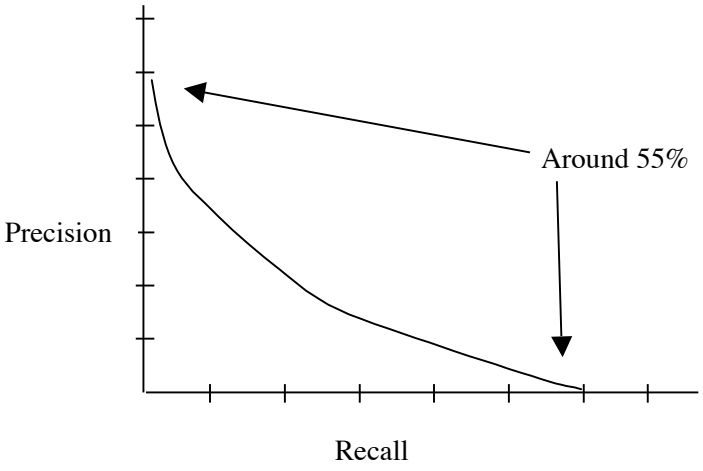
Precision = [# docs retrieved and correct] / [# docs retrieved]

Recall = [# docs retrieved and correct] / [# total docs correct in corpus]

F-score = P.R / [βR + (1 - β)P] where β = relative importance of P to R

Typical tradeoff. Best systems today: between 40% and 66% R and P, depending on genre and query length, etc.

Typical P/R curve:



The TREC competitions (http://trec.nist.gov/) were started around 1990 by Donna Harman at the National Institute of Standards and Technology (NIST), near Washington DC. Initially, the competition simply provided a uniform test for straightforward IR engines, but pretty soon it grew other ''tracks': cross-language IR, video IR, interactive IR, topic spotting, one for question answering, etc. Its highly successful formula both advanced research and helped people share results. About five years ago, researchers in Europe organized the CLEF conferences, doing the same thing, and focusing much more on the cross-language problem. Japan followed around 2000, with the NTCIR series (*http://research.nii.ac.jp/ntcir/*), that included summarization.

4. Cross-lingual IR

Want to retrieve documents in a foreign language. Obviously you don't want to translate the whole foreign corpus into your language! But then you have to translate the query somehow. Problems:

- word ambiguity across the language barrier: things get worse! Especially when you don't have a full sentence, just a few words. The meaning fanout problem.
- varying transliterations of names in foreign scripts (Clinton = *Kurinton* in Japanese)

Solutions:

- use simple bilingual wordlists: either most common translation or all
- maintain part of speech
- use MT engine to translate query
- try semantic-level translation of query (e.g., Dorr and Oard with Dorr's Lexical-Conceptual Structure)
- try to disambiguate by word family relatedness, etc.
- "lexical triangulation"

(Dorr and Oard 1998): 57,780 Spanish newspaper articles with 25 queries from TREC-4, using Amherst's Inquery system. They find

Human translation	P= 0.249
MT of query	P= 0.141
Dictionary, keeping all translations for each query word	P= 0.137
Dictionary, keeping most frequent translation	P= 0.136
Lexical-conceptual (semantic) structure for query	P= 0.105
Dictionary, keeping all translations, but stemmed	P= 0.093
No translation (hope for identical foreign spelling)	P= 0.023

In recent years, however, people have refined search so that there is essentially no more penalty when you cross the language barrier. The TREC-2001 results even show that cross-lingual IR outperforms monolingual IR in some (unexplained) cases.

5. Text Clustering

Sometimes you want to return sets of documents, grouping similar ones together. The problem is to characterize each document in a compressed but accurate way and then cluster together related documents, so you can then treat the cluster as a single macro-document.

Also work in the word vector space. Use some **distance measure**—most common are the cosine (distance) coefficient, the Dice, and the Jaccard coefficients.

Agglomerative clustering proceeds step by step:

- compute distance between every pair of clusters
- merge the closest two clusters into one mini-cluster
- recur until done.

The result is a dendogram—a binary tree structure, with the leaves being the individual document points, the internal nodes being (partial) clusters, and the arcs recording the distance between any two (partial) clusters or documents.

What does 'closest' mean, when the cluster has some kind of irregular shape in the space? You have to specify a **policy of grouping**—to decide which of the possible points of a cluster to measure:

- between closest points of two clusters: SLINK (result: long sausage-shaped clusters)
- between further points of two clusters: CLINK (result: balanced blobs)

- between median points of two clusters: MEDIAN (result balanced trees)
- so that internal variance of new cluster is minimized: Ward's Method (result: somewhat balanced trees)

Computing the distance from every cluster to every other cluster in every loop of the cycle is expensive. A speedup method is call Reciprocal Nearest Neighbor (RNN). At the start of new cluster step, do not compare *every* two points in the space to find next cluster to form; choose a point/cluster at random, find its closest point/cluster, go to that one, find the closest point./cluster to the new one, and so on. Stop when the closest one to your current cluster is the one you have just come from; you have reached a local minimum. Then cluster these two together there.

There are other methods as well: k-nearest neighbors (kNN), sparse matrix diagonalization, Latent Semantic Indexing. Different methods on the same data give very different results; this is not an exact science.

When you have the dendogram, you still have a problem of deciding where the actual clusters are (i.e., how to chop up the dendogram). There are three traditional **cluster delimiting policies**: chop at large enough inter-node distance scores, or chop into clusters of appropriate size, or chop into a specified number of clusters.

Clustering has never been satisfactorily accurate. The **major problem** is that these clusters are formed on the basis of word sharing across documents. But 'true' semantic clustering cares more about word *relatedness*, like in a thesaurus: ideally, you would use a vector space whose dimensions are word meanings (expressed as words, phrases, or even pictures). You can approximate this with synonym sets and word families (topic signatures).

6. Question Answering

Newest TREC addition: instead of retrieving the appropriate document(s) from a collection of them, retrieve the appropriate sentence fragment(s) from a collection of sentences in texts.

Input a simple NL query ("who discovered America?" "how long does it take to fly from Paris to New York in the Concorde?" "where is the Taj Mahal?" "how old is the President?"); output a string of text shorter than 50 bytes.

Q&A combines IR with Information Extraction (template-filling) techniques. Question and answer typologies.

See the lecture next week.

7. The Web

Some (recent) search engines: Lycos, AltaVista (with starter ontology), Yahoo!, Google, etc. For web search, 'spidering' (searching for documents and storing their URLs) is a relatively new issue in IR. Spiders are run every day/night, and build massive index structures. Some files are visited daily, others weekly, others less often; the scheduling of revisits is an issue in itself. No search engine visits all files; by current estimates, the Web contains between 100 and 150 terabytes of text in 2004, and Google spiders only between 9 and 20 terabytes.

Google ranks its documents by the number of pointers to them, and reranks by the number of visits a document has received. That's why a famous person's personal website is often at the top when you search for him/her, even though another page may mention this person more often.

Some engines use a manually created ontology (Yahoo!'s has a subject hierarchy; Google uses OpenDirectory, which is uncontrolled), some use just vector space. A few years ago, everyone had the idea of extending WordNet and using it to enrich queries (by synonym expansion). One such company, first called Oingo and then AppliedSemantics (here in Santa Monica), eventually had an ontology of some 300,000 terms. They were bought by Google in late 2003.

A big problem is the trustworthiness / quality of a webpage—what to do? Three strategies:

- Count number of pointers to it from other pages
- Consider how other pager refer to it (using scare words or not)
- Scan page for scare words ("really great", "very", "extremely", "wonderful") or 'serious' words ("estimate", "research has shown", "likely that", "authority"), etc.

None of these is infallible, but they each provide contributory evidence.

8. Current IR Systems

Research systems: Smart (Cornell); Inquery (UMass Amherst); SabIR (small company using Smart); MG (opensource: free from Sydney at http://www.mds.rmit.edu.au/mg/).

Optional further readings:

Overview books:

- Baeza-Yaes, R. and B. Ribiero-Neto. 1999. *Modern Information Retrieval.* Addison Wesley.
- Frakes, W.B. and R. Baeza-Yates. 1992. *Information Retrieval*. Prentice Hall.
- Browne, M. and M. Berry. 1999. *Understanding Search Engines: Mathematical Modelling and Text Retrieval*. SIAM Series on Software, Environment, and Tools.
- Duda, Hart, Stork. Pattern Classification.

Technical papers:

- *SIGIR Conference Proceedings*, every year.
- *TREC Proceedings*, published by NIST, every year.

Evaluation:

- Harman, D. 1998. *Proceedings of the LREC Conference*, 517.

Cross-lingual IR:

- Dorr, B. and D. Oard. 1998. *Proceedings of the LREC Conference*, 759.
- *TREC Proceedings*, 2000, 2001.

Other issues:

- Topic Signatures: Lin, C.-Y. and E.H. Hovy. 2000. *COLING Proceedings*.

Web search

- Brin, S. and L. Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *WWW7 / Computer Networks* 30(1–7), 107–117.