Information Retrieval Methods

Helena Ahonen-Myka
Spring 2007, part 4
Indexing (2/2)
Translation from Finnish: Greger Lindén

In this part

- Making a term more narrow
  - Constructing phrases
- Making a term broader
  - Using a thesaurus
- Constructing an inverted file and using it

2

Making a term more narrow: constructing phrases

- If a sequence of words (a phrase) has some meaning, this meaning is always more narrow than the single words in it
  - "computer science" vs. "computer"
- If phrases are added to the document description, the intension is usually to narrow down the meaning of some terms that are too broad
  - Goal: terms with a high frequency are changed to terms with average frequency
  - Two rare terms should not be combined, because the phrase would be even more rare

3

Possible algorithm

- The head word in a phrase is a word,
  - whose document frequency exceeds a certain threshold (e.g. df > 5) or
  - whose discrimination value is negative
- Other components of the phrase are rare or average terms that occur in the context of the head word (e.g. in the same sentence close enough to the head word)
- Stopwords are usually not included as parts of phrases, at least not in the beginning or end of the phrase
  - in some cases stopwords make a difference: "flights to London", "flights from London"

4

Choosing components for a phrase

- Terms other than the head word can be chosen in many ways
  - That is, ”occur in the context of” can be interpreted in many ways
- Let us look at the following example:
  - "Effective retrieval systems are essential for people in need of information."

5

Choosing components for a phrase

- The terms ”are”, ”for”, ”in”, and ”of” are probably stopwords and are not taken into account
- The terms ”systems”, ”people” and ”information” are probably frequent enough to fit as the head words of phrases
- If we require that the head word and one other component are subsequent, we get as phrases
  - retrieval systems, systems essential, essential people, people need, need information

6

Choosing components for a phrase

- If it is enough that the components are in the same sentence, we get the additional phrases
  - effective systems, systems need, effective people, retrieval people, effective information, retrieval information, essential information
- Maintaining the order of the terms is usually worthwhile
- We can also put additional constraints on the head word and the components
  - If we know the part of speech for the words, we could accept only e.g. adjective-noun or noun-noun pairs

7

Choosing components for a phrase

- If we are able to parse the syntactic structure of the text, we can require that the components in a phrase are included in the same syntactic/functional component, e.g., in a subject phrase, a verb phrase or an object phrase
- Syntactic phrases in the example:
  - Subject phrase: effective retrieval systems
  - Verb phrase: are essential
  - Object phrase: people in need of information
- We would accept the following phrases: effective systems, retrieval systems, people need, need information

8

Choosing components for a phrase

- With tighter constraints we will produce fewer phrases
- Both loose and tight constraints may produce both good and bad phrases
- We could continue and try to find out the semantic relationships between words
  - "high frequency transistor oscillator": high frequency is ok, frequency transistor is not
  - It might be difficult and laborious and may not improve the results significantly
- "wrong" phrases may also help in matching between queries and documents

9

Variation in phrase structures

- Because of matching methods, we should be able to merge phrases that mean the same thing but are different (syntactically)
  - "information retrieval" vs. "retrieval of information"
  - synonyms, different word orders, fillers
- We can try to normalise phrases into some canonical form
- Or construct alternative phrases of each original phrase
- Both alternatives are in practice quite troublesome

10

Example on variations

- Basic form: text analysis system
- variations:
  - System analyses the text
  - Text is analysed by the system
  - System carries out text analysis
  - Text is subjected to analysis by the system
  - Text is subjected to system analysis
- Synonyms that could replace terms
  - text → documents, information items
  - analysis → processing, transformation, manipulation
  - system → program, process

11

Finding phrases

- Instead of extracting phrases directly from documents (as above), phrases can be found in many ways
  - Common phrases in search logs
  - Using heuristic rules for special types of phrases (e.g. patterns for names of people or companies)

12

Making a term broader: using a thesaurus

- A term that occurs too rarely can be replaced by a more general term
  - A more general term can be found in a conceptual model (thesaurus, ontology)
- A thesaurus groups narrow terms into classes
  - The combined occurrence frequency of the members in the groups are on average level
  - E.g. 'refusal', 'declining', 'non-compliance', 'rejection', and 'denial' could belong to the same group
  - Occurrences of the group members in a document can be replaced by a group identifier, which can be one of the members (e.g. 'refusal')

13

Constructing a thesaurus

- We can construct a thesaurus either automatically or manually
- Manual thesauruses are e.g..
  - WordNet: a general thesaurus in English
  - Topical thesaurus in some particular field
- Manual work can be supported by automatic methods, e.g., we can automatically produce lists of
  - all occurrences of a word in the collection → the word may take on different meanings in different contexts
  - different terms occurring in similar environments → the terms belong to the same group

14

Constructing a thesaurus

- Automatic methods
  - We compare the co-occurrence of terms
  - We use a set of retrieval tasks and associated relevance evaluations

15

Document-term matrix

	T1	T2	...	Tt
D1	w_{11}	w_{12}	...	w_{1t}
D2	w_{21}	w_{22}	...	w_{2t}
.	.			.
.	.			.
.	.			.
Dn	w_{n1}	w_{n2}	...	w_{nt}

16

Co-occurrence of terms

- We want to find terms that occur frequently together
- The similarity between two terms may be denoted by the following similarity measure

$$sim(T_j, T_k) = \sum_{i=1}^{N} w_{ij} \cdot w_{ik}$$

- Where N is the number of documents
- When we have computed pair-wise similarity values, we can cluster terms that are similar into the same groups

17

Clustering terms

- There are several ways to cluster, for instance
  - A term is added to a cluster if the similarity value of the term with at least one member of the cluster exceeds a given threshold
    - This method usually produces fewer and larger clusters
  - A term is added to a cluster if the similarity values of the term and all the terms in the cluster exceed a given threshold
    - This method usually produces much smaller clusters
- Terms in a cluster form a thesaurus group

18

Using retrievals and relevance estimates

- We assume that we can use a document collection, a set of retrieval tasks, and their corresponding relevance estimates
- We assume that term T_j occurs in the query Q and another term T_k in the document D, which is relevant for the query Q
- If T_j and T_k are grouped in the same thesaurus group, the similarity between Q and D will increase (which is desirable)
  - T_j and T_k are also replaced by the same group identifier in the documents
- We can also make sure that the thesaurus groups do not contain two terms where one occurs in a retrieval task and the other in a document that is non-relevant for the task

Constructing a thesaurus automatically

- If we use an automatically constructed thesaurus, we can use it only to replace terms when indexing the same kinds of texts
  - Or otherwise we have to use very diverse texts so that the groups that we obtain are general enough
- If we take retrieval tasks and relevance estimates into account, the tasks must also cover the different topics of the collection very well

Summary: constructing descriptions for documents

- Collect all the words that occur in a document
- Remove stopwords
- Modify the remaining words, if needed
- Compute weights for terms in all documents using the tf·idf function
- Describe the document with a set of terms and their weights $D_i = \{T_1, w_{i1}; T_2, w_{i2}; ...; T_t, w_{it}\}$

Alternative method

- Collect all the words that occur in a document
- Remove stopwords
- Modify the remaining words, if needed
- Compute a discrimination value for all terms
- Replace all terms with a discrimination value close to zero (i.e. very rare terms) with more general terms, with the help of a thesaurus
- Replace terms with a negative discrimination value (i.e. very common terms) with phrases

Alternative, cont.

- Compute weights for single terms, phrases and concepts of the thesaurus
  - The weight of a phrase is e.g. the average weight of the components
- Describe each document with a set of single terms, phrases, and thesaurus groups, as well as corresponding weights
- In both alternatives we can say that in the collection there are T terms and each document is described with these T terms
  - If a term does not occur in a document, its weight is zero

Descriptions for queries

- If queries are given in natural language, their descriptions are formed just as in the case of documents
  - terms + weights
- Because queries are usually short, the term frequency (tf) does not have any significance
  - As weight we use only the inverse document frequency (idf)

Constructing an index

- After selecting a set of terms and computing their weights, we have a stored set of terms (in a sequential file) for each document
- A query contains a set of terms
  - In a retrieval task we have to find the documents where the terms occur quickly
- We construct an inverted file where for each term we have the documents in which the term occurs
- In addition, we have a dictionary file as an index for the inverted file

25

dictionary file:

terms	a	b	c	d	e	f	g	h	j	k	l	...
df	3	2	3	3	1	2	3	1	3	2	2	

inverted file:

terms	a	b	c	d	e	f	g	h	j	k	l	...
documents	136	37	246	689	8	48	137	2	157	59	24	

base file:

documents	1	2	3	4	5	6	7	8	9	...		...
terms	agj	cht	abg	cft	jk	acd	bgj	def	dk	...		

26

Constructing an inverted file

- An inverted file can be constructed in several different ways, e.g.,
- The base file is read one document at a time
  - We construct a list of (term, document) pairs
    - (a,1), (g,1), (j,1), (c,2), (h,2), (t,2), (a,3), (b,3), (g,3), (c,4), (f,4), (t,4), (j,5), (k,5), (a,6), (c,6), (d,6),…
- The list is ordered in ascending order of the terms (if same term, in order of the document number)
    - (a,1), (a,3), (a,6), (b,3), (c,2), (c,4), (c,6), (d,6), (f,4), (g,1), (g,3), (h,2), (j,1), (j,5), (k,5), (t,2), (t,4),…

27

Constructing an inverted file

- At the end we combine pairs with the same term: we add all document numbers to the same term in an ordered list
  - (a,<1,3,5>), (b,<3>), (c,<2,4,6>), (d, <6>), (f,<4>), (g, <1,3>), (h, <2>), (j,<1,5>), (k, <5>), (t,<2,4>),…
  - From this representation we can also form the dictionary file
- The list of document numbers for a term are also called **postings**

28

About implementation

- In the previous example, we left out the term weights in the documents; but they are also considered to be in the base file
  - We pick triplets (term, document, term weight)
  - If we use tf · idf weights, it is enough to store the tfs because the idf of a term is the same in all documents
  - (That is: idf can be computed from the dictionary file; tf can be computed from the inverted file)

29

About implementation

- In this method, the most expensive operation is sorting the (term, document) pairs
- When the document collection is fairly big, sorting cannot be made in main memory
  - But it can be done by external merge sort

30

Sorting

- We assume that
  - The (term, document) pairs are stored on disk
  - The main memory can hold k (term, document) pairs at once
- We read k (term, document) pairs into the main memory and sort them with e.g. quicksort
- The ordered list is written back onto the disk
- We repeat this until all pairs have been sorted once (all lists of k pairs sorted)

31

Sorting

- Merging:
  - We read the first two lists from the disk and merge them into one list and write them back onto the disk
  - We read the next two lists etc. and continue until all lists of length k have been processed
  - Then we read the first two lists of length 2k and merge them, etc.
  - We continue until there is only one list left

32

Performance

- The more pairs that can fit into the main memory, the faster the indexing method is
- The method requires a lot of disk space
  - During the sorting we need two copies of the file containing the pairs
- Very large collections must be sorted with other methods

33

Index granularity

- In the previous slides, we stored information about the positions of the terms on the accuracy level of a document
- If we want to support proximity queries or return text fragments smaller than a whole document, we can mark positions more accurately in the index
- We could also define a document to be a text paragraph, a sentence or a word
  - But we would lose information about the hierarchy of the components in a document

34

Index granularity

- The ”normal” case:
  - information: <D345, D348,D350,…>
  - retrieval: <D123, D128, D345,…>
- We add information about in which sentence a term occurs:
  - information: <D345,25; D345,37; D348, 10; D350,8;…>
  - retrieval: <D123,5; D128,25; D345,37; D345,40;…>
  - We can quickly answer the query : “’information’ in same sentence as ‘retrieval’” → D345
  - More space is required due to two reasons: 1) the sentence number and 2) terms that occur more than once in documents produce more elements in the list (before only one element)

35

Index granularity

- We could also add information about the position of a term in the sentences
  - information: <D345,25,4; D345,37,3; D348, …>
  - retrieval: <D123,5,2; D128,25,4; D345,37,4;…>
  - We can answer queries like
    - “information adjacent to retrieval”
    - “information and retrieval within five words”
- If we have also stored meta data about documents (author, title, publisher, …) we can add information to the index about the position in some meta data
  - We can answer queries like “the author is John Irving”

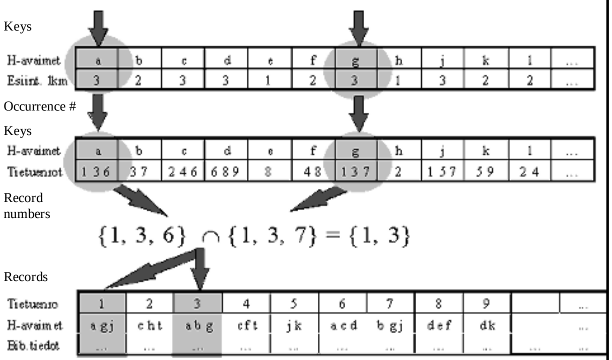
36

Index granularity

- If we do not expect to make many queries using proximity (nearness) operators, indexing on the document level is enough
  - Proximity constraints can be checked from the answer sets in the postprocessing phase

Using an inverted file

- We fetch the records from the dictionary corresponding to the search words in the query
  - The records hold pointers to the corresponding records in the inverted file
- We fetch the records corresponding to the terms from the inverted file
  - These records hold lists of (pointers) to the documents where the terms occur
- We use the document lists according to the format of the query
  - E.g.. ”a and g”: we find documents that are on the lists of terms a and g
- We fetch the documents based on their numbers from the base file
- If we were not able to solve all conditions with the help of the inverted file, we scan the documents and check the remaining conditions



Keys and bib files

In this part

- Constructing phrases on more general terms
- Replacing terms with more general concepts from a thesaurus
- Constructing a thesaurus automatically
- Constructing an inverted file by sorting
- Index granularity
- Using an inverted file