Modern Information Retrieval

Chapter 7: Text Processing

Fall 2011

Overview

1. Document pre-processing
    1. Lexical analysis
    2. Stopword elimination
    3. Stemming
    4. Index-term selection
    5. Thesauri
2. Text Compression
    1. Statistical methods
    2. Huffman coding
    3. Dictionary methods
    4. Ziv-Lempel compression

Document Preprocessing

- Document pre-processing is the process of incorporating a new document into an information retrieval system.
- The goal is to
  - Represent the document *efficiently* in terms of both *space* (for storing the document) and *time* (for processing retrieval requests) requirements.
  - Maintain good *retrieval performance* (precision and recall).
- Document pre-processing is a complex process that leads to the representation of each document by a select set of *index* terms.
- However, some Web search engines are giving up on much of this process and index *all* (or virtually all) the words in a document).

Document Preprocessing (cont.)

- Document pre-processing includes 5 stages:
  1. Lexical analysis
  2. Stopword elimination
  3. Stemming
  4. Index-term selection
  5. Construction of thesauri

Lexical analysis

- *Objective*: Determine the words of the document.
- Lexical analysis separates the input alphabet into
  - Word characters (e.g., the letters a-z)
  - Word separators (e.g., space, newline, tab)
- The following decisions may have impact on retrieval
  - *Digits*: Used to be ignored, but the trend now is to identify numbers (e.g., telephone numbers) and mixed strings as words.
  - *Punctuation marks*: Usually treated as word separators.
  - *Hyphens*: Should we interpret "pre-processing" as "pre processing" or as "preprocessing"?
  - Letter case: Often ignored, but then a search for "First Bank" (a specific bank) would retrieve a document with the phrase "Bank of America was the first bank to offer its customers…"

Stopword elimination

- *Objective*: Filter out words that occur in most of the documents.
- Such words have no value for retrieval purposes
- These words are referred to as stopwords. They include
  - Articles (a, an, the, …)
  - Prepositions (in, on, of, …)
  - Conjunctions (and, or, but, if, …)
  - Pronouns (I, you, them, it…)
  - Possibly some verbs, nouns, adverbs, adjectives (make, thing, similar, …)
- A typical stopword list may include several hundred words.
- As seen earlier, the 100 most frequent words add-up to about 50% of the words in a document.
- Hence, stopword elimination improves the size of the indexing structures.

Stemming

- *Objective*: Replace all the *variants* of a word with the single *stem* of the word.
- Variants include plurals, gerund forms (ing-form), third person suffixes, past tense suffixes, etc.
- *Example*: **connect**: connects, connected, connecting, connection,…
- All have similar semantics and relate to a single concept.
- In parallel, stemming must be performed on the user query.

Stemming (cont.)

- Stemming improves
  - *Storage and search efficiency*: less terms are stored.
  - *Recall*:
    - without stemming a query about “connection”, matches only documents that have “connection”.
    - With stemming, the query is about “connect” and matches *in addition* documents that originally had “connects”, “connected”, “connecting”, etc.
- However, stemming may hurt *precision*, because users can no longer target just a particular form.
- Stemming may be performed using
  - *Algorithms* that stripe of suffixes according to substitution rules.
  - Large *dictionaries*, that provide the stem of each word.

Index term selection (indexing)

- *Objective*: Increase efficiency by extracting from the resulting document a *selected set of terms* to be used for indexing the document.
  - If full text representation is adopted then *all* words are used for indexing.
- Indexing is a critical process: User's ability to find documents on a particular subject is limited by the indexing process having created index terms for this subject.
- Index can be done *manually* or *automatically*.
- Historically, manual indexing was performed by professional indexers associated with library organizations.
- However, automatic indexing is more common now (or, with full text representations, indexing is altogether avoided).

Indexing (cont.)

- Relative advantages of manual indexing:
  - Ability to perform *abstractions* (conclude what the subject is) and determine additional *related* terms,
  - Ability to judge the *value* of concepts.
- Relative advantages of automatic indexing:
  - Reduced cost: Once initial hardware cost is amortized, operational cost is cheaper than wages for human indexers.
  - Reduced processing time
  - Improved consistency.
- *Controlled vocabulary*: Index terms must be selected from a predefined set of terms (the *domain* of the index).
  - Use of a controlled vocabulary helps standardize the choice of terms.
  - Searching is improved, because users know the vocabulary being used.
  - Thesauri can compensate for lack of controlled vocabularies.

Indexing (cont.)

- *Index exhaustivity*: the extent to which concepts are indexed.
  - Should we index only the most important concepts, or also more minor concepts?
- *Index specificity*: the preciseness of the index term used.
  - Should we use general indexing terms or more specific terms?
  - Should we use the term "computer", "personal computer", or “Gateway E-3400”?
- Main effect:
  - High exhaustivity improves recall (decreases precision).
  - High specificity improves precision (decreases recall).
- Related issues:
  - Index title and abstract only, or the entire document?
  - Should index terms be weighted?

Indexing (cont.)

Reducing the *size* of the index:

- Recall that articles, prepositions, conjunctions, pronouns have already been removed through a stopword list.
  - Recall that the 100 most frequent words account for 50% of all word occurrences.
- Words that are *very infrequent* (occur only a few times in a collection) are often removed, under the assumption that they would probably not be in the user's vocabulary.
- Reduction not based on probabilistic arguments: *Nouns* are often preferred over verbs, adjectives, or adverbs.

Indexing (cont.)

Indexing may also assign *weights* to terms.

- *Non-weighted indexing*:
  - No attempt to determine the value of the different terms assigned to a document.
  - Not possible to distinguish between major topics and casual references.
  - All retrieved documents are equal in value.
  - Typical of commercial systems through the 1980s.
- *Weighted indexing*:
  - Attempt made to place a value on each term as a description of the document.
  - This value is related to the frequency of occurrence of the term in the document (higher is better), but also to the number of collection documents that use this term (lower is better).
  - Query weights and document weights are combined to a value describing the likelihood that a document matches a query

Thesauri

Objective: Standardize the index terms that were selected.

- In its simplest form a thesaurus is
  - A list of "important" words (concepts).
  - For each word, an associated list of synonyms.
- A thesaurus may be generic (cover all of English) or concentrate on a particular domain of knowledge.
- The role of a thesaurus in information retrieval
  - Provide a standard vocabulary for indexing.
  - Help users locate proper query terms.
  - Provide hierarchies for automatic broadening or narrowing of queries.
- Here, our interest is in providing a standard vocabulary (a controlled vocabulary).
- Essentially, in this final stage, each indexing term is *replaced* by the concept that defines its thesaurus class.

Text Compression

- *Data Encoding*: Transform encoding units (characters, words, etc.) into code values.
  - *Objective* is either
    - Reduce size (compression)
    - Hide contents (encryption).
- *Lossless encoding*: The transformation is *reversible*– original file can be recovered from encoded (compressed, encrypted) file.
- *Compression ratio*:
  - *S*: size of the uncompressed file.
  - *C*: size of the compressed file.
  - *Compression-rate = C/S.*
  - *Example*:
    - S= 300,000 bytes, C=100,000 bytes.
    - Compression rate: 100,000/300,000 = 0.33.

Text Compression (cont.)

- *Advantages* of compression:
  - Reduction in storage size.
  - Reduction in transmission time.
  - Reduction in processing times (e.g., searching).
- Disadvantages:
  - Requires time for compression/decompression.
  - Processing of compressed text is more complex.
- Specific for information retrieval:
  - Decompression time is often more critical than compression time.
    - Unlike transmission-motivated compression (modems), documents in an information retrieval system are encoded once and decoded many times.
  - Prefer compression techniques that allow searching in the compressed file (without decompressing it).

Text compression (cont.)

Basic methods:

- Statistical methods:
  - Estimate the probability of occurrence of each encoding unit (character or word) in the alphabet.
  - Assign codes to units: more frequent units are assigned shorter codes.
  - In information retrieval, word-encoding is preferred over character encoding.
- Dictionary methods:
  - Substitute a *phrase* (string of units) by a pointer to a dictionary or a previous occurrence of the phrase.
  - Compression is achieved because the pointer is shorter than the phrase.

Statistical methods

- Recall from the discussion of information theory:
  - Assume a message from an alphabet of *n* symbols.
  - Assume that the probability of the *i*'th symbol is *pi*.
  - The average information content (*entropy*) is:

$$E = -\sum_{i=1}^{n} pi.\log 2(pi)$$

  - *Optimal encoding* is achieved when a symbol with probability *pi* is assigned a code whose length is log2(1/*pi*) = –log2(*pi*) .
  - Hence, *E* also represents *optimal average code length* (measured in bits per character).
  - Therefore, *E* is the *lower bound* on compression.

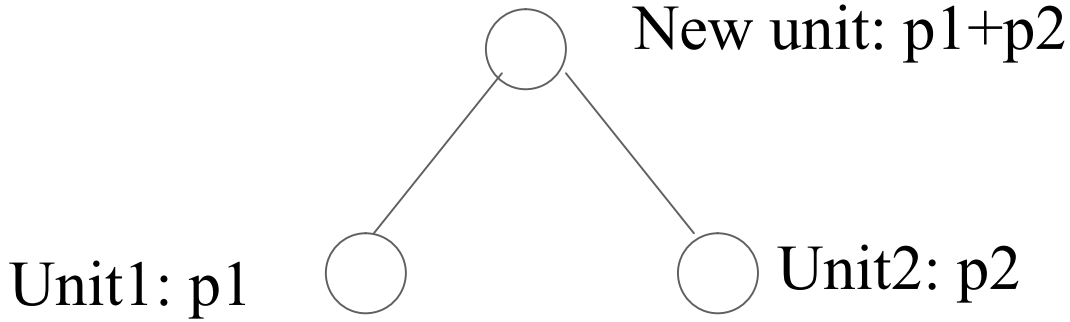
Statistical methods (cont.)

- Statistical methods must first estimate the frequencies of the encoding units, and then assign codes based on these frequencies.
- Approaches:
  - *Static*: Use a single distribution for all texts.
    - Fast, but not optimal because different texts exhibit different distributions.
    - The encoding table is stored in the application (not in the text).
    - Decompression can start at any point in the file.
  - *Dynamic*: Determine the frequencies in a preliminary pass.
    - Excellent compression, but a total of two passes is required.
    - The encoding table is stored at the beginning of the text.
    - Decompression can start at any point in the file.
  - *Adaptive*: Progressively learn the distribution of the text while compressing; each character is encoded on the basis of the preceding characters in a text.
    - Fast, and close to optimal compression.
    - Decompression must start from the beginning

Huffman coding

- *General*:
  - Huffman coding is one of the best known compression techniques (1952).
  - It is used in the Unix programs pack/unpack.
  - It is a statistical method based on variable length codes.
  - Compression is achieved by assigning shorter codes to more frequent units.
  - Decompression is unique because no code is the prefix of another.
  - Encoding units may be either bytes or words.
  - Does not exploit the *dependencies* between the encoding units.
  - Yields *optimum* average code length when these units are independent.
  - Can be used with the static, dynamic and adaptive approaches.

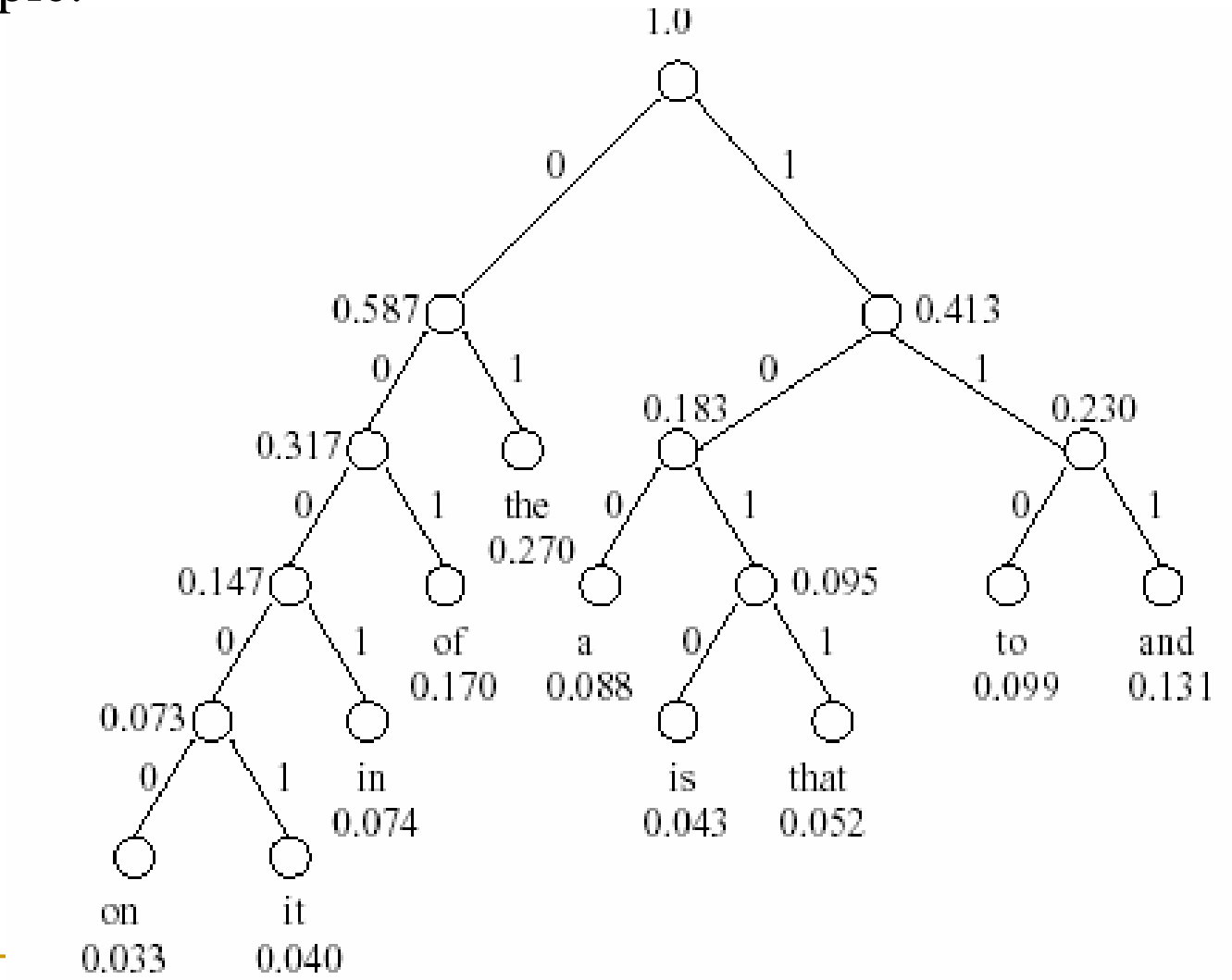
Huffman coding (cont.)

- *Method*:
  - 1. Build a table of the encoding units and their frequencies (probabilities).
  - 2. Combine the two *least* frequent units into a unit with the *sum* of the probabilities and encode it in a new "unit".
  - 

  - 3. Repeat this process until the entire dictionary is represented by a root whose probability is 1.0.
  - 4. When there is a *tie* for the two least frequent units, any tie-breaking procedure is acceptable.

Huffman coding (cont.)

Example:



Huffman coding (cont.)

- *Example (cont.)*:
  - The resulting code:
  - Average code length:

$$\sum_{i=1}^{10} pi.li = 3.05bits$$

  - The entropy (compression lower bound) is:

$$\sum_{i=1}^{10} pi.\log 2(pi) = 3.01bits$$

  - Fixed code length would have required log2 10 = 3.32 bits (which, in practice, would require 4 bits).
  - Compression ratio:

C/S = 3.05/3.32 = 0.92

Encoding unit	Code value	Code length (l_i)
the	01	2
of	001	3
and	111	3
to	110	3
a	100	3
in	0001	4
that	1011	4
is	1010	4
it	00001	5
on	00000	5

Huffman coding (cont.)

- *Example:* When the letters A-Z are thus encoded:
  - Code lengths are between 3 and 10 bits.
  - Average code length is 4.12 bits.
  - A fixed code would have required log2 26 = 4.70 bits (i.e., 5 bits).
- More compression is obtained by encoding *words*:
  - With the 800 most frequent English words (small table!) are encoded in this method (all other words are in plain ASCII), 40-50% compression has been reported.
- Huffman codes are prefix-specific:
  - No code is the beginning of another code.
  - Hence, a left-to-right decoding operation is *unique*.
  - It is possible to search the compressed text.

Dictionary methods

- Dictionary methods construct a dictionary of phrases, and replace their occurrences with dictionary pointers.
- The choice of phrases may be static, dynamic or adaptive.
- A simple method (digrams):
  - Construct a dictionary of *pairs* of letters that occur together frequently (e.g., ou, ea, ch, …).
  - If *n* such pairs are used, a pointer (location in the dictionary) requires log2 *n* bits.
  - At each step in the encoding, the next pair is examined.
    - If it corresponds to a dictionary pair, it is replaced by its encoding, and the encoding position moves by 2 characters.
    - Otherwise, the single character encoding is kept, and the position moves by one character.
  - To assure that decoding is unambiguous, an extra bit is needed to indicate whether the next unit is a single character code or a digram code.

Ziv-Lempel compression

- *General*:
  - The Ziv-Lempel method (1977) uses a single-pass adaptive scheme.
  - While compressing, it constructs a dictionary from phrases encountered so far.
  - Many popular programs (Unix compress/uncompress, GNU gzip/gunzip, and Windows WinZip) are based on the Ziv-Lempel algorithm.
  - Compression is slightly better than Huffman codes (C/S of 45% vs. 55%).
  - Disadvantage for information retrieval: decompressed file cannot be searched and decoding cannot start at a random place in the file.

Ziv-Lempel compression (cont.)

- *Compression:*
  - 1. Initialize the dictionary to contain all “phrases” of length one.
  - 2. Examine the input stream and search for the longest phrase which has appeared in the dictionary.
  - 3. Encode this phrase by its index in the dictionary.
  - 4. Add the phrase followed by the next symbol in the input stream to the dictionary.
  - 5. Go to Step 2.

Ziv-Lempel compression (cont.)

Example:

- Assume a dictionary of 16 phrases (4 bit index).

Data	a	b	b	a	a	b	b	a	a	b	a	b	b	a	a	a	a	b	a	a	b	b	a	b	a
Encryption	0	1	1	0	2		4		2		6			5		5		7			3		8		

- This case does not result in compression.
  - *Source*: 25 characters in a 2-character alphabet require a total of 25 bits.
    - *Output*: 13 pointers of 4 bits require a total of 52 bits.
- This is because the length of the input data in this example is too short.
- In practice, the Lempel-Ziv algorithm works well only when the input data is sufficiently large and there is sufficient redundancy in the data.

Dictionary			
Index	Entry	Index	Entry
0	a	8	aba
1	b	9	abba
2	ab	10	aaa
3	bb	11	aab
4	ba	12	baab
5	aa	13	bba
6	abb	14	
7	baa	15	