PV030 Textual Information Systems

Petr Sojka

Faculty of Informatics
Masaryk University, Brno

Spring 2012

Indexing methods

- manual vs. automatic, pros/cons
- ***stop-list*** (words with grammatical meaning – conjunctions, prepositions, ...)
  1. not-driven
  2. driven (a special dictionary of words: indexing language assessment) – ***pass-list***, thesaurus.
- synonyms and related words.
- inflective languages: creating of registry with language support – ***lemmatization***.

Text analysis – choice of words for index

Frequency of word occurrences is for document identification significant.

English frequency dictionary:

1	the	69971	0.070
2	of	36411	0.073
3	and	28852	0.086
4	to	26149	0.104
5	a	23237	0.116
6	in	21341	0.128
7	that	10595	0.074
8	is	10099	0.088
9	was	9816	0.088
10	he	9543	0.095

- ***Zipf's law*** (principle of least resistance)
  order × frequency ≅ constant
- ***Cumulative proportion of used words*** $CPW = \frac{\sum_{order=1}^{N} frequency_{order}}{text\ words\ count}$
- The rule 20–80: 20 % of the most frequent words make 80 % of text [MEL, fig. 4.19].

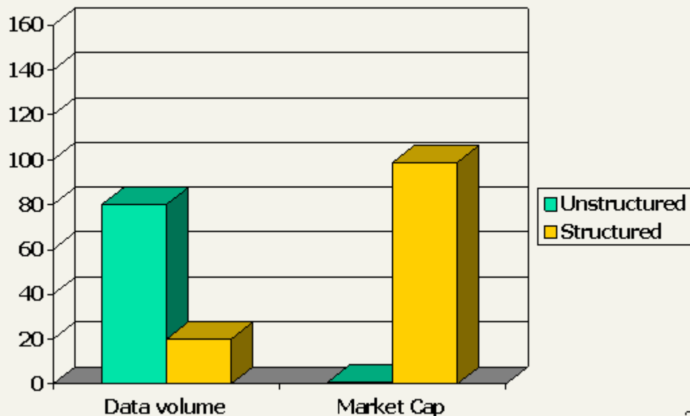
Automatic indexing method

Automatic indexing method is based on word significance derivation from word frequencies (cf. Collins-Cobuild dictionary); words with low and high frequency are cut out:
INPUT: n documents
OUTPUT: a list of words suitable for an index creation

1. We calculate a frequency $FREQ_{ik}$ for every document $i \in \langle 1, n \rangle$ and every word $k \in \langle 1, K \rangle$ [K is a count of different words in all documents].
2. We calculate $TOTFREQ_k = \sum_{i=1}^{n} FREQ_{ik}$.
3. We create a frequency dictionary for the words $k \in \langle 1, K \rangle$.
4. We set down a threshold for an exclusion of very frequent words.
5. We set down a threshold for an exclusion of words with a low frequency.
6. We insert the remaining words to the index.

Questions of threshold determination [MEL, fig. 4.20].

Unstructured (text) vs. structured (database) data in 1996



Unstructured (text) vs. structured (database) data in 2006

160
140
120
100
80
60
40
20
0

Data volume

Market Cap

Google

YAHOO!

Unstructured
Structured

IR using the Boolean model

- Queries are Boolean expressions, *e.g.*, Caesar AND Brutus
- The seach engine returns all documents that satisfy the Boolean expression
- Does Google use the Boolean model?

Unstructured data in 1650

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***?
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return)
    - Later lectures

Term-document incidence

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 if play contains word, 0 otherwise

Brutus** AND **Caesar** but NOT **Calpurnia

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) → bitwise *AND*.
- 110100 *AND* 110111 *AND* 101111 = 100100.

Answers to query

- **Antony and Cleopatra, Act III, Scene ii**
- *Agrippa* [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
- When Antony found Julius ***Caesar*** dead,
- He cried almost to roaring; and he wept
- When at Philippi he found ***Brutus*** slain.

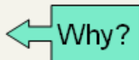
- **Hamlet, Act III, Scene ii**
- *Lord Polonius:* I did enact Julius ***Caesar*** I was killed i' the
- Capitol; ***Brutus*** killed me.

Bigger corpora

- Consider N = 1M documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data in the documents.
- Say there are m = 500K ***distinct*** terms among these.

Can't build the matrix

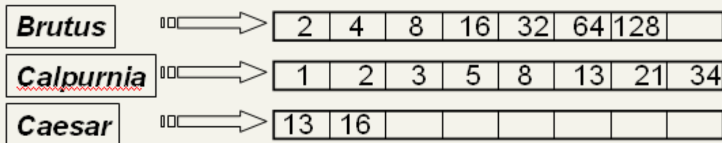
- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's.
  - matrix is extremely sparse.



- What's a better representation?
  - We only record the 1 positions.

Inverted index

- For each term *T*, we must store a list of all documents that contain *T*.
- Do we use an array or a list for this?



What happens if the word ***Caesar*** is added to document 14?

Inverted index construction

Documents to be indexed.

Friends, Romans, countrymen.

⋮

Tokenizer

Token stream.

Friends | Romans | Countrymen

More on these later.

Linguistic modules

Modified tokens.

friend | roman | countryman

Indexer

Inverted index.

friend ⇒ 2 → 4 →

roman ⇒ 1 → 2 →

countryman ⇒ 13 → 16

12

Indexer steps

- Sequence of (Modified token, Document ID) pairs.

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

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

- Sort by terms.

Core indexing step.

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

→

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

- Multiple term entries in a single document are merged.
- Frequency information is added.

Why frequency?
Will discuss later.

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Term	Doc #	Term freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

term	docID	freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

⟹

term	coll. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	3	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	2	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	2	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

- The result is split into a *Dictionary* file and a *Postings* file.

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

→

Term	N docs	Coll freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

- Where do we pay in storage?

Term	N docs	Coll freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	
2	
1	
1	
2	
1	
1	
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

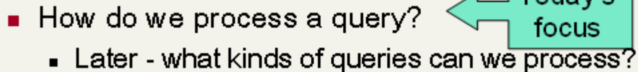
Terms

Pointers

Will quantify the storage, later.

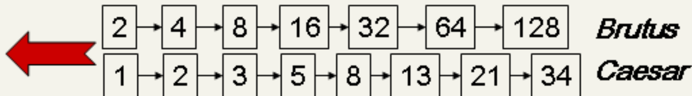
The index we just built

- How do we process a query?
  - Later - what kinds of queries can we process?



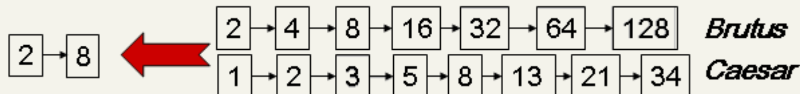
Query processing: AND

- Consider processing the query:
  Brutus *AND* ***Caesar***
  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are x and y, the merge takes $O(x+y)$ operations.

Crucial: postings sorted by docID.

Intersecting ("merging") two postings lists

```
MERGE(p, q)
1  answer ← ⟨ ⟩
2  while p ≠ NIL and q ≠ NIL
3  do if docID[p] = docID[q]
4        then ADD(answer, docID[p])
5        else  if docID[p] < docID[q]
6                 then p ← next[p]
7                 else  q ← next[q]
8  return answer
```

Boolean queries: Exact match

- The Boolean Retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not.
- Primary commercial retrieval tool for 3 decades.
- Professional searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you're getting.

Example: WestLaw http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- Tens of terabytes of data; 700,000 users
- Majority of users *still* use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
- /3 = within 3 words, /S = in same sentence

Example: WestLaw http://www.westlaw.com/

- Another example query:
  - Requirements for disabled people to be able to access a workplace
  - disabl! /p access! /s work-site work-place (employment /3 place
- Note that SPACE is disjunction, not conjunction!
- Long, precise queries; proximity operators; incrementally developed; not like web search
- Professional searchers often like Boolean search:
  - Precision, transparency and control
- But that doesn't mean they actually work better….

Boolean queries: More general merges

- Exercise: Adapt the merge for the queries:

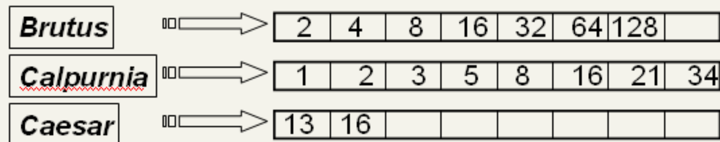
  Brutus AND NOT Caesar

  Brutus OR NOT Caesar

Can we still run through the merge in time $O(x+y)$ or what can we achieve?

Query optimization

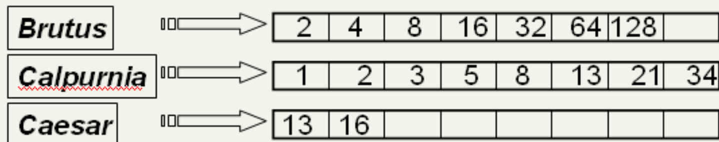
- What is the best order for query processing?
- Consider a query that is an *AND* of *t* terms.
- For each of the *t* terms, get its postings, then *AND* them together.



Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Query optimization example

- Process in order of increasing freq:
  - *start with smallest set, then keep cutting further.*



Execute the query as (***Caesar*** *AND* ***Brutus****)* *AND* ***Calpurnia***.

Optimized intersection of a set of postings lists

MERGE($\langle t_i \rangle$)
1 *terms* ← SORTBYFREQ($\langle t_i \rangle$)
2 *result* ← *postings*[*first*[*terms*]]
3 *terms* ← *rest*[*terms*]
4 **while** *terms* ≠ NIL and *result* ≠ NIL
5 **do** *list* ← *postings*[*first*[*terms*]]
6 *terms* ← *rest*[*terms*]
7 MERGEINPLACE(*result*, *list*)
8 **return** *result*

More general optimization

- e.g., *(**madding** OR **crowd**) AND (**ignoble** OR **strife**)*
- Get freq's for all terms.
- Estimate the size of each *OR* by the sum of its freq's (conservative).
- Process in increasing order of *OR* sizes.

Exercise

- Recommend a query processing order for

(tangerine OR trees) AND
(marmalade OR skies) AND
(kaleidoscope OR eyes)

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Exercise

- Try the search feature at http://www.rhymezone.com/shakespeare/
- Write down five search features you think it could do better

What's ahead in IR? Beyond term search

- What about phrases?
  - ***Stanford University***
- Proximity: Find ***Gates*** *NEAR* ***Microsoft***.
  - Need index to capture position information in docs. More later.
- Zones in documents: Find documents with (*author* = ***Ullman***) *AND* (text contains ***automata***).

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Often we want to rank/group results
  - Need to measure proximity from query to each doc.
  - Need to decide whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

Evidence accumulation

- 1 vs. 0 occurrence of a search term
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
  - Usually more seems better
- Need term frequency information in docs

IR vs. databases: Structured vs unstructured data

- Structured data tends to refer to information in "tables"

Employee	Manager	Salary
Smith	Jones	50000
Chang	Smith	60000
Ivy	Smith	50000

Typically allows numerical range and exact match (for text) queries, e.g.,
Salary < 60000 AND Manager = Smith.

The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users, queries, information needs
- Beyond terms, exploit ideas from social networks
  - link analysis, clickstreams ...

- How do search engines work? And how can we make them better?

Resources for today's lecture

- Introduction to Information Retrieval, ch. 1
- Managing Gigabytes, Chapter 3.2
- Modern Information Retrieval, Chapter 8.2
- Shakespeare: http://www.rhymezone.com/shakespeare/
  - Try the neat browse by keyword sequence feature!

Any questions?

Outline (Week seven)

- Excursus to the computational linguistics.
- Corpus linguistics as an TIS example.
- Search methods with preprocessing of text and pattern (query).

Lemmatization for index creation

Morphology utilization for creating of dictionary

- ☞ stem/ root of words (učit, uč);
- ☞ program ajka (abin), `http://nlp.fi.muni.cz/projekty/ajka/` examples;
- ☞ a techniques of patterns for stem determination;

Registry creating – thesaurus

- Thesaurus – a dictionary, containing hierarchical and associative relations and relations of equivalence between particular terms.
- Relations between terms/lemmas:
  - ***synonyms*** – relation to a standard term; *e.g.* „see";
  - relation to a related term (RT); *e.g.* „see also";
  - relation to a broader term (BT);
  - relation to a narrower term (NT);
  - ***hypernyms*** (car:means of transport); ***hyponyms*** (bird:jay); ***meronym*** (door:lock); ***holonyms*** (hand:body); ***antonyms*** (good:bad).
- Dog/Fík, Havel/president

Thesaurus construction

manually/ half-automatically

- heuristics of thesaurus construction:
  - hierarchical structure/s of thesaurus
  - field thesauri, the semantics is context-dependent (e.g. field, tree in informatics)
  - compounding of terms with a similar frequency
  - exclusion of terms with a high frequency
- breadth of application of thesaurus and lemmatizer: besides of spelling indexing, base of grammar checker, fulltext search.
- projekts WORDNET, EUROWORDNET
- `module add wordnet; wn`
  `wn faculty -over -simsn -coorn`

Hierarchical thesaurus

- Knowledge base creation for exact evaluation of document relevance.
- ***topic*** – processing of semantic maps of terms Visual Thesaurus `http://www.visualthesaurus.com`.
- Tovek Tools, Verity.

Part I

Excursus to the Computational Linguistics

Computational linguistics

- ☞ string searching – words are strings of letters.
- ☞ word-forming – morphological analysis.
- ☞ grammar (CFG, DFG) – syntactic analysis.
- ☞ meaning of sentences (TIL) – semantic analysis.
- ☞ context – pragmatic analysis.
- ☞ full understanding and communication ability – information.

Corpus Query Processor

basic queries

- „Havel“;

45:	Český	prezident	Václav	<Havel>	se	včera	na
89:	jak	řekl	Václav	<Havel>	,	každý	občan
248:	více	než	rokem	<Havel>	řekl	Pravda	vítězí

regular expressions

- „Pravda|pravda“;
- „(P|p)ravda“;
- „(P|p)ravd[a,u,o,y]“;
- „pravd.*“; „pravd.+“; „post?el“;

word sequence

- „prezident(a|u)“ „Havl(a|ovi)“;
- „a tak“;
- „prezident“; []* „Havel“;
- „prezident“ („republiky“ „Vaclav“)? „Havel“;

Corpus Query Processor

queries for positional attributes

- [word = „Havel"];
- [lemma = „prezident"] []* [lemma = „Havel"];
- … ženu prezidenta Havla …
  [lemma = „hnát"] [] [lemma = „Havel"];
- [word = „žen(u|eme)" & lemma !=„žena"]; | … *or*
  ! … not

some other possibilities

- [lemma = „prezident"] []* [lemma = „Havel"] within s; …10, 3 s
- [lemma = „Havel"] within 20 </s>„Pravda"
- <s>a:[word= „Žena|Muž|Člověk"] []* [lemma = a.lemma]

Face and back of relevant searching

Large computational power of today's computers enables:

- efficient storing of large amount of text data (compression, indexing);
- efficient search for text strings.

A man sitting behind a computer uses all this, to obtain from so processed documents information, that he is interested. Really?

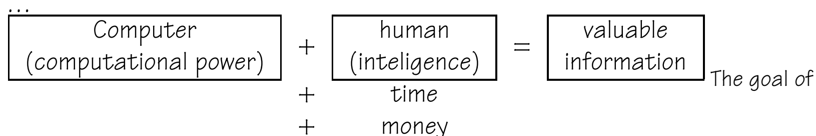
Example: In text database there is stored a few last years of daily newspaper. I'd like to obtain information about president Václav Havel.

a/>HAVEL

b/>more precise queries

c/...

...



The goal of everybody → is to transfer the largest possible part of intelligence (time, money, ...) to computer.

Face and back of relevant searching

information	ideal of ideals	no	Searching	
pragmatic analysis	context	no	information	Correct
semantic analysis	sentence meaning TIL	starting-up	Spell	translation
syntactic analysis	grammar CFG, DCG	partially	check	
morphological analysis	word-forming lemma	yes	Check	Simple translat
words are strings of letters	string searching	yes		

Face and back of data acquisition from natural language

Do we really know, what is information contained in the text in natural language?

- František Novák weighs 100 kg. ⟶ **RDB**
  object property value — attribut1, attribut2, …
- František Novák likes beer. ? key value
  František Novák likes ↗
  Jana Novotná
- F. N. is an old honest man. ⟶ ?
  Spring erupted in full force.

Words of the natural language denote objects, their properties and relations between them. It's possible to see the words and sentences also as „functions“ of its kind, defined by their meaning.

- A man, who climbed a highest Czech eight-thousander, is my grandson.

Corpus linguistics

- ☞ ***Corpus***: electronic collection of texts, often indexed by linguistic tags.
- ☞ Corpus as a text information system: corpus linguistics.
- ☞ BNC, Penn Treebank, DESAM, PNK, . . . ; ranges from millions to billion positions (words), special methods necessary.
- ☞ Corpus managers CQP, GCQP, Manatee/Bonito, `http://www.fi.muni.cz/~pary/`

see [MAR].

What's a corpus?

Definition: ***Corpus*** is a large, internaly structured compact file of texts in natural language electronically stored and processable.

- Indian languages have no script – for a finding of a grammar it's necessary to write up the spoken word.
- 1967 – 1. corpus in U. S. A. (Kučera, Francis) 1 000 000 words.
- Noam Chomsky – refuses corpora.
- Today – massive expansion.

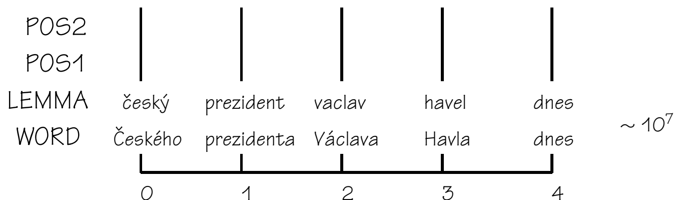
Corpora on FI

- WWW page of Pavel Rychlý (~pary) links to basic information. Bonito, Manatee.
- IMS CORPUS WORKBENCH – a toolkit for efficient representation and querying over large text files.

Logical view of corpus

Sequence of words at numbered positions (first word, *n*th word), to which ***tags*** are added (addition of tags called corpus ***tagging***). Tags are morphological, grammatical and any other information about a given word. It leads to more general concept of ***position attributes***, those are the most important tagging type. Attributes of this class have a value (string) at every corpus position. To every of them one word is linked as a basic and positional attribute word. In addition to this attribute, further position attributes may be bundled with each position of any text, representing the morphological and other tags.

Structural attributes – sentences, paragraphs, title, article, SGML.



Internal architecture of corpus

Two key terms of internal representation of position attributes are:

- **Uniform representation:** items for all attributes are encoded as integer numbers, where the same values have the same digital code. A sequence of items is then represented as a sequence of integers. Internal representation of attribute word (as well as of any other pos. attribute) is `array(0..p-1) of Integer`, where `p` is position count of corpus.
- **Inverted file:** for a sequence of numbers representing a sequence of values of a given attribute, the inverted file is created. This file contains a set of occurrences in position attribute for every value (better value code). Inverted file is needed for searching, because it directly shows a set of occurrences of a given item, the occurrences then can be counted in one step.

Internal architecture of corpus (cont.)

File with encoded attribute values and inverted file as well have auxiliary files.

- The first data structure is a **list of items** or „lexicon“: it contains a set of different values. Internally it's a set of strings occurring in the sequence of items, where a symbol `Null` (octal 000) is inserted behind every word. The list of items already defines a code for every item, because we suppose the first item in the list to have a code 0, following 1 etc.

Internal architecture of corpus (cont.)

There are three data structures for the inverted file:

- The first is an independent inverted file, that contains a set of corpus positions.
- The second is an index of this file. This index returns for every code of item an input point belonging to an occurrence in inverted file.
- The third is a table of frequency of item code, which for each item code gives a number of code occurrence in corpus (that is of course the same as the size of occurrence set).

Recap of the previous lecture

- Basic inverted indexes:
  - Structure: Dictionary and Postings
  - Key step in construction: Sorting
- Boolean query processing
  - Simple optimization
  - Linear time merging
- Overview of course topics

Plan for this lecture

Finish basic indexing

- The Dictionary
  - Tokenization
  - What terms do we put in the index?
- Postings
  - Query processing – faster merges
  - Proximity/phrase queries

Recall basic indexing pipeline

Documents to be indexed.

Friends, Romans, countrymen.

⋮

Tokenizer

Token stream.

Friends	Romans	Countrymen

Linguistic modules

Modified tokens.

friend	roman	countryman

Indexer

Inverted index.

friend → 2 → 4

roman → 1 → 2

countryman → 13 → 16

Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

Each of these is a classification problem, which we will study later in the course.

But these tasks are often done heuristically …

Complications: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages.
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
- What is a unit document?
  - A file?
  - An email? (Perhaps one of many in an mbox.)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX in HTML)

Tokenization

Definitions

- **Word** – A string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – A class of all tokens containing the same character sequence.

Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** →

    Finland? Finlands? Finland's?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***State-of-the-art***: break up hyphenated sequence.
    - ***co-education*** ?
    - ***the hold-him-back-and-drag-him-away-maneuver*** ?
    - It's effective to get the user to put in possible hyphens
  - ***San Francisco***: one token or two? How do you decide it is one token?

Numbers

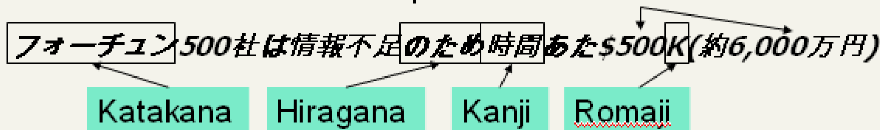
- ***3/12/91*** ***Mar. 12, 1991***
- ***55 B.C.***
- ***B-52***
- ***My PGP key is 324a3df234cb23e***
- ***100.2.86.144***
  - Often, don't index as text.
    - But often very useful: think about things like looking up error codes/stacktraces on the web
    - (One answer is using n-grams: Lecture 3)
  - Will often index "meta-data" separately
    - Creation date, format, etc.

Tokenization: Language issues

- ***L'ensemble*** → one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***
- German noun compounds are not segmented
  - Lebensversicherungsgesellschaftsangestellter
  - 'life insurance company employee'

Tokenization: language issues

- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Arabic script example

كِتَابٌ ⇐ ٌ ب ا ت ِ ك

un b ā t i k

/kitābun/ *'a book'*

Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures
- استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.
- ← → ← → ← start
- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’
- With Unicode, the surface presentation is complex, but the stored form is straightforward

Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form
  - We want to match ***U.S.A.*** and ***USA***
- We most commonly implicitly define equivalence classes of terms
  - e.g., by deleting periods in a term
- Alternative is to do asymmetric expansion:
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient

Normalization: other languages

- Accents: ***résumé*** vs. ***resume***.
- Most important criterion:
  - How are your users like to write their queries for these words?

- Even in languages that standardly have accents, users often may not type them

- German: Tuebingen vs. Tübingen
  - Should be equivalent

Normalization: other languages

- Need to "normalize" indexed text as well as query terms into the same form

  7月30日 vs. 7/30

- Character-level alphabet detection and conversion
  - Tokenization not separable from this.
  - Sometimes ambiguous:

    Morgen will ich in MIT …

Is this German "mit"?

Case folding

- Reduce all letters to lower case
  - exception: upper case (in mid-sentence?)
    - e.g., *General Motors*
    - *Fed* vs. *fed*
    - *SAIL* vs. *sail*
  - Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization…

Stop words

- With a stop list, you exclude from dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - They take a lot of space: ~30% of postings for top 30
- But the trend is away from doing this:
  - Good compression techniques (lecture 5) means the space for including stopwords in a system is very small
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: "King of Denmark"
    - Various song titles, etc.: "Let it be", "To be or not to be"
    - "Relational" queries: "flights to London"

Thesauri and soundex

- Handle synonyms and homonyms
  - Hand-constructed equivalence classes
    - e.g., ***car*** = ***automobile***
    - ***color*** = ***colour***
- Rewrite to form equivalence classes
- Index such equivalences
  - When the document contains ***automobile***, index it under ***car*** as well (usually, also vice-versa)
- Or expand query?
  - When the query contains ***automobile***, look under ***car*** as well

Soundex

- Traditional class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for names
  - E.g., ***chebyshev*** → ***tchebycheff***
- More on this later ...

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form

Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggest crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter's algorithm

- Commonest algorithm for stemming English
  - Results suggest at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- *(m>1) EMENT* →
  - *replacement* → *replac*
  - *cement* → *cement*

Other stemmers

- Other stemmers exist, e.g., Lovins stemmer
  http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
  - Single-pass, longest suffix removal (about 250 rules)
  - Motivated by linguistics as well as IR
- Full morphological analysis – at most modest benefits for retrieval
- Do stemming and other normalizations help?
  - Often very mixed results: really help recall for some queries but harm precision on others

Does stemming improve effectiveness?

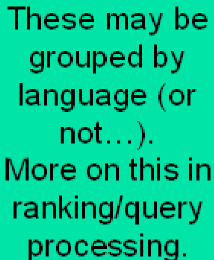
- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Examples: "operational AND research", "operating AND system", "operative AND dentistry"
- Porter Stemmer equivalence class ("oper"): operate operating operates operation operative operatives operational

Language-specificity

- Many of the above features embody transformations that are
  - Language-specific and
  - Often, application-specific
- These are "plug-in" addenda to the indexing process
- Both open source and commercial plug-ins available for handling these

Dictionary entries – first cut

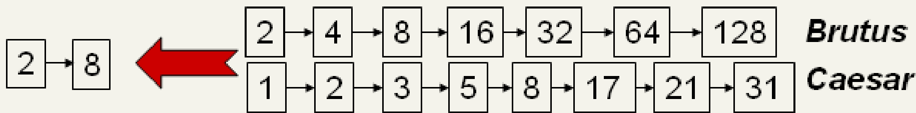
ensemble.french
時間.japanese
MIT.english
mit.german
guaranteed.english
entries.english
sometimes.english
tokenization.english



Faster postings mergers: Skip pointers

Recall basic merge

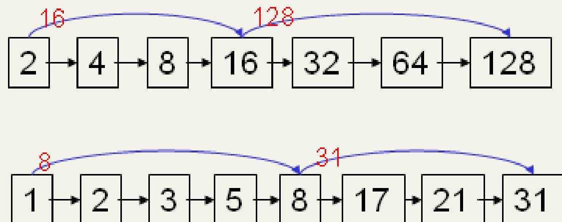
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are m and n, the merge takes $O(m+n)$ operations.

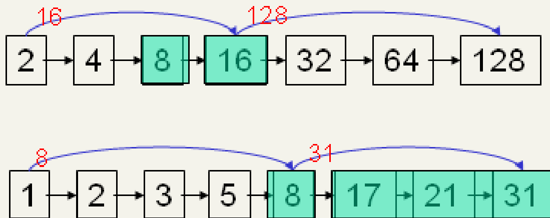
Can we do better?
Yes, if index isn't changing too fast.

Augment postings with skip pointers (at indexing time)



- Why?
- To skip postings that will not figure in the search results.
- How?
- Where do we place skip pointers?

Query processing with skip pointers



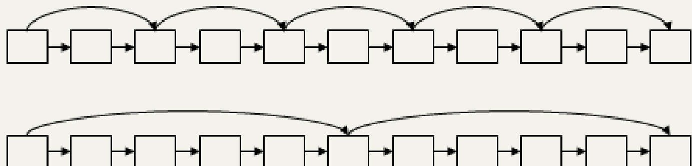
Suppose we've stepped through the lists until we process **8** on each list.

When we get to **16** on the top list, we see that its successor is **32**.

But the skip successor of **8** on the lower list is **31**, so we can skip ahead past the intervening postings.

Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.



Placing skips

- Simple heuristic: for postings of length L, use $\sqrt{L}$ evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder if L keeps changing because of updates.

- This definitely used to help; with modern hardware it may not (Bahle et al. 2002)
  - The cost of loading a bigger postings list outweighs the gain from quicker in memory merging

Phrase queries

Phrase queries

- Want to answer queries such as "***stanford university***" – as a phrase
- Thus the sentence *"I went to university at Stanford"* is not a match.
  - The concept of phrase queries has proven easily understood by users; about 10% of web queries are phrase queries
- No longer suffices to store only

<*term* : *docs*> entries

A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text "Friends, Romans, Countrymen" would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

Longer phrase queries

- Longer phrases are processed as we did with wild-cards:
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university *AND* ***university palo*** *AND* ***palo alto***

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Can have false positives!

Extended biwords

- Parse the indexed text and perform part-of-speech-tagging (POST).
- Bucket the terms into (say) Nouns (N) and articles/prepositions (X).
- Now deem any string of terms of the form NX*N to be an extended biword.
  - Each such extended biword is now made a term in the dictionary.
- Example: ***catcher in the rye***

  N X X N
- Query processing: parse it into N's and X's
  - Segment query into enhanced biwords
  - Look up index

Issues for biword indexes

- False positives, as noted before
- Index blowup due to bigger dictionary

- For extended biword index, parsing longer queries into conjunctions:
  - E.g., the query ***tangerine trees and marmalade skies*** is parsed into
  - ***tangerine trees*** *AND* ***trees and marmalade*** *AND* ***marmalade skies***

- Not standard solution (for all biwords)

Solution 2: Positional indexes

- Store, for each ***term***, entries of the form:

  <number of docs containing ***term***;

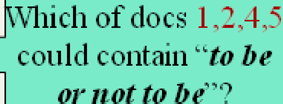
  doc1: position1, position2 … ;

  doc2: position1, position2 … ;

  etc.>

Positional index example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>



- Can compress position values/offsets
- Nevertheless, this expands postings storage *substantially*

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to***:
    - 2:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be***:
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
- Same general method for proximity searches

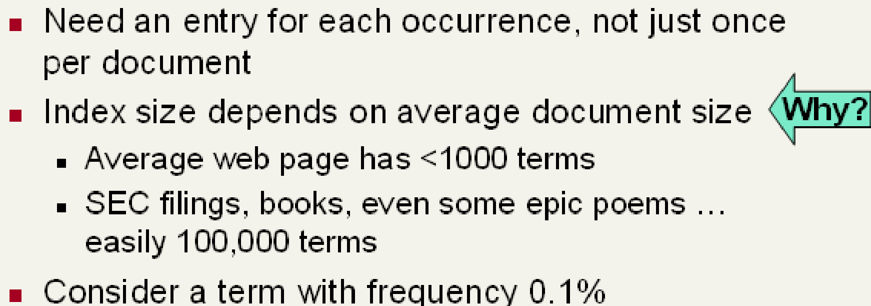
Proximity queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  Here, /*k* means "within *k* words of".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of *k*?

Positional index size

- You can compress position values/offsets: we'll talk bout that in lecture 5
- Nevertheless, a positional index expands postings storage *substantially*
- Nevertheless, it is now standardly used because of the power and usefulness of phrase and proximity queries … whether used explicitly or implicitly in a ranking retrieval system.

Positional index size



Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Rules of thumb

- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of volume of original text
- Caveat: all of this holds for “English-like” languages

Combination schemes

- These two approaches can be profitably combined
  - For particular phrases (***"Michael Jackson", "Britney Spears"***) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like "***The Who***"
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - A typical web query mixture was executed in ¼ of the time of using just a positional index
  - It required 26% more space than having a positional index alone

Resources for today's lecture

- IIR 2
- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer: http://www.tartarus.org/~martin/PorterStemmer/
- Skip Lists theory: Pugh (1990)
  - Multilevel skip lists give same O(log *n*) efficiency as trees
- H.E. Williams, J. Zobel, and D. Bahle. 2004. "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.

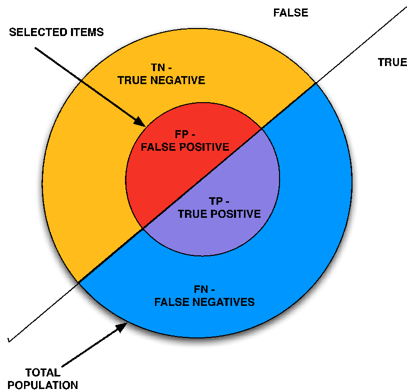
  http://www.seg.rmit.edu.au/research/research.php?author=4
- D. Bahle, H. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. SIGIR 2002, pp. 215-221.

Search methods IV.

Preprocessing of text and pattern (query): *overwhelming majority of* today's TIS. Types of preprocessing:

- *n*-gram statistics (fragment indexes).
- special algorithms for indexes processing (coding, compression) and relevance evaluation (PageRank Google)
- usage of natural language processing methods (morphology, syntactic analysis, semantic databases) an aggregation of information from multiple sources (systems AnswerBus, START).
- signature methods.

Sensitivity



$$\text{Accuracy} = \frac{tp+tn}{tp+fp+fn+tn}$$

$$\text{Recall (Sensitivity)} = \frac{tp}{tp+fn}$$

$$\text{Precision (Positive Predictive Value)} = \frac{tp}{tp+fp}$$

$$\text{False Positive Rate} = \frac{fp}{fp+tn}$$

$$\text{False Negative Rate} = \frac{fn}{tp+fn}$$

$$\text{Specificity} = \frac{tn}{tn+fp}$$

$$\text{Negative Predictive Value} = \frac{tn}{tn+fn}$$

Relevance

Definition: ***Relevance*** (of answers to a query) is a rate range, by which a selected document coincides with requirements imposed on it.
Ideal answer $\equiv$ real answer
Definition: ***Coefficient of completeness (recall)*** $R = \frac{m}{n}$, where m is a count of selected relevant records and n is a count of all relevant records in TIS.
Definition: ***Coefficient of precision*** $P = \frac{m}{o}$, where o is count of all selected records by a query.
We want to achieve maximum R and P, tradeoff.

Standard values: 80 % for P, 20 % for R.
Combination of completeness and precision:
coefficient $F_b = \frac{(b^2+1)PR}{b^2P+R}$. ($F_0 = P$, $F_\infty = R$, where $F_1 = F$ P and R weighted equally).

Fragment index

- ☞ The fragment ybd is in English only in the word molybdenum.
- ☞ Advantages: fixed dictionary, no problems with updates.
- ☞ Disadvantages: language dependency and thematic area, decreased precision of search.

Outline (Week ten)

- Google as an example of web-scale information system.
- Jeff Dean's video – historical notes of Google search developments.
- Google – system architecture.
- Google – PageRank.
- Google File System.
- Implementation of index systems

Goooooooooooooogle – a bit of history

An example of anatomy of global (hyper)text information system (www.google.com).

- 1997: google.stanford.edu, students Page and Brin
- 1998: one of few quality search engines, whose basic fundamentals and architecture (or at least their principles) are known – therefore a more detailed analysis according to the article [G00]
  http://www7.conf.au/programme/fullpapers/1921com1921.htm.
- 2012: clear leader in global web search

Goooooooooooooogle – anatomy

- Several innovative concepts: PageRank, storing of local compressed archive, calculation of relevance from texts of hypertext links, PDF indexing and other formats, Google File System, Google Link...
- The system anatomy. see [MAR]

Google: Relevance

The crucial thing is documents' relevance (credit) computation.

- ☞ Usage of tags of text and web typography for the relevance calculation of document terms.
- ☞ Usage of text of hyperlink is referring to the document.

Google: PageRank

- **PageRank**: objective measure of page importance based on citation analysis (suitable for ordering of answers for queries, namely page relevance computation).
- Let pages $T_1,\ldots,T_n$ (citations) point to a page A, total sum of pages is m. PageRank

$$PR(A) = \frac{(1-d)}{m} + d\left(\frac{PR(T_1)}{C(T_1)} + \ldots \frac{PR(T_n)}{C(T_n)}\right)$$

- PageRank can be calculated by a simple iterative algorithm (for tens of millions of pages in hours on a normal PC).
- PageRank is a probability distribution over web pages.
- PageRank is not the only applied factor, but coefficient of more factors. A motivation with a random surfer, dumping factor d, usually around 0.85.

Data structures of Google

- Storing of file signatures
- Storing of lexicon
- Storing of hit list.
- Google File System

Index system implementation

- Inverted file – indexing file with a bit vector.
- Usage of document list to every key word.
- Coordinate system with pointers [MEL, fig. 4.18, page 46].
- Indexing of corpus texts: Finlib **http://www.fi.muni.cz/~pary/dis.pdf** *see* [MAR].
- Use of Elias coding for a compression of hit list.

Index system implementation (cont.)

- Efficient storing of index/dictionary [lemmas]: ***packed trie***, Patricia tree, and other tree structures.
- Syntactic neural network (S. M. Lucas: Rapid best-first retrieval from massive dictionaries, Pattern Recognition Letters 17, p. 1507–1512, 1996).
- Commercial implementations: *Verity engine*, *most of web search engines* – with few *exceptions* – hide their key to success.

Dictionary representation by FA I

Article M. Mohri: On Some Applications of Finite-State Automata Theory to Natural Language Processing see [MAR]

- Dictionary representation by finite automaton.
- Ambiguities, unification of minimized deterministic automata.
- Example: *done,do.V3:PP*
  done,done.A0
- Morphological dictionary as a list of pairs [word form, lemma].
- Compaction of storing of data structure of automata (Liang, 1983).
- Compression ratio up to 1:20 in the linear approach (given the length of word).

Dictionary representation by FA II

- Transducer for dictionary representation.
- Deterministic transducer with 1 output (subsequential transducer) for dictionary representation including one string on output (information about morphology, hyphenation,...).
- Deterministic transducer with p outputs (p–subsequential transducer) for dictionary representation including more strings on output (ambiguities).
- Determinization of the transducer generally unrealizable (the class of deterministic transducers with an output is a proper subclass of nondeterministic transducers); for purposes of natural language processing, though, usually doesn't occur (there aren't cycles).

Dictionary representation by FA III

- An addition of a state to a transducer corresponding (w_1,w_2) without breaking the deterministic property: first a state for (w_1,ε), then with resulting state final state with output w_2.
- Efficient method, quick, however not minimal; there are minimizing algorithms, that lead to spatially economical solutions.
- Procedure: splitting of dictionary, creation of det. transducers with p outputs, their minimization, then a deterministic unification of transducers and minimizing the resulting.
- Another use also for the efficient indexing, speech recognition, etc.

Part II

Coding

Outline (Week eleven)

- Coding.
- Entropy, redundancy.
- Universal coding of the integers.
- Huffman coding.
- Adaptive Huffman coding.

Coding – basic concepts

Definition: ***Alphabet A*** is a finite nonempty set of symbols.
Definition: ***Word*** (***string***, ***message***) over A is a sequence of symbols from A.
Definition: ***Empty string ε*** is an empty sequence of symbols. A set of nonempty words over A is labeled A^+.
Definition: ***Code*** K is a triad (S, C, f), where S is finite set of ***source units***, C is finite set of ***code units***, $f : S \longrightarrow C^+$ is an injective mapping. f can be expanded to $S^+ \longrightarrow C^+$: $F(S_1 S_2 \ldots S_k) = f(S_1)f(S_2)\ldots f(S_k)$.
C^+ is sometimes called ***code***.

Basic properties of the code

Definition: $x \in C^+$ is ***uniquely decodable*** regarding f, if there is maximum *one* sequence $y \in S^+$ so, that $f(y) = x$.

Definition: Code $K = (S, C, f)$ is ***uniquely decodable*** if all strings in C^+ are uniquely decodable.

Definition: A code is called a ***prefix*** one, if no code word is a prefix of another.

Definition: A code is called a ***suffix*** one, if no code word is a suffix of another.

Definition: A code is called a ***affix*** one, if it is prefix and suffix code.

Definition: A code is called a ***full*** one, if after adding of any additional code word a code arises, that isn't uniquely decodable.

Basic properties of code

Definition: ***Block code of length n*** is such a *code*, in which all *code words* have length n.

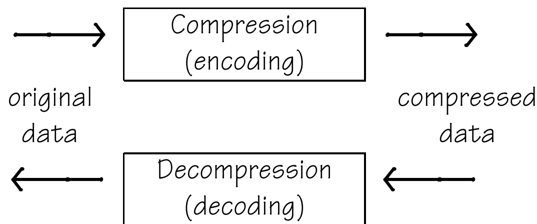
Example: block ? prefix

block $\Rightarrow$ prefix, but not vice versa.

Definition: A *code* $K = (S, C, f)$ is called ***binary***, if $|C| = 2$.

Compression and decompression

Definition: ***Compression*** (***coding***), ***decompression*** (***decoding***):



Definition: ***Compression ratio*** is a ratio of length of compressed data and length of original data.

Example: Suggest a binary prefix code for decimal digits, if there are often numbers 3 a 4, and rarely 5 and 6.

Entropy and redundancy I

Let Y be a random variable with a probability distribution $p(y) = P(Y = y)$. Then the mathematical *expectation* (mean rate) $E(Y) = \sum_{y \in Y} y p(y)$.

Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of source units and let the occurrence probability of unit x_i in information source **S** is p_i for $i = 1, \ldots, n$, $n \in N$.

Definition: ***Entropy of information content of unit*** $\mathbf{x_i}$ (measure of amount of information or uncertainty) is $H(x_i) = H_i = -\log_2 p_i$ bits.

A source unit with more probability bears less information.

Entropy and redundancy II

Definition: ***Entropy of information source*** $\mathbf{S}$ is $H(\mathbf{S}) = -\sum_{i=1}^{n} p_i \log_2 p_i$ bits.

True, that $H(\mathbf{S}) = \sum_{y \in Y} p(y) \log \frac{1}{p(y)} = E\left(\log \frac{1}{p(Y)}\right)$.

Definition: ***Entropy of source message*** $X = x_{i_1} x_{i_2} \ldots x_{i_k} \in \mathbf{S}^+$ ***of information source*** $\mathbf{S}$ is $H(X, \mathbf{S}) = H(X) = \sum_{j=1}^{k} H_i = -\sum_{j=1}^{k} \log_2 p_{i_j}$ bits.

Definition: ***Length*** $l(X)$ ***of encoded message*** X

$l(X) = \sum_{j=1}^{k} |f(x_{i_j})| = \sum_{j=1}^{k} d_{i_j}$ bits.

Theorem: $l(X) \geq H(X, \mathbf{S})$.

Entropy a redundancy III

Axiomatic introduction of entropy see [MAR], details of derivation see `ftp://www.math.muni.cz/pub/math/people/Paseka/lectures/kodovani.ps`

Definition: $R(X) = I(X) - H(X) = \sum_{j=1}^{k} (d_{i_j} + \log_2 p_{i_j})$ is ***redundancy of code K for message X***.

Definition: ***Average length of code word K*** is $AL(K) = \sum_{i=1}^{n} p_i d_i$ bits.

Definition: ***Average length of source S*** is

$$AE(\mathbf{S}) = \sum_{i=1}^{n} p_i H_i = -\sum_{i=1}^{n} p_i \log_2 p_i \text{ bits.}$$

Definition: ***Average redundancy of code K*** is

$$AR(K) = AL(K) - AE(\mathbf{S}) = \sum_{i=1}^{n} p_i (d_i + \log_2 p_i) \text{ bits.}$$

Entropy and redundancy IV

Definition: A *code* is an ***optimal*** *one*, if it has minimal redundancy.

Definition: A *code* is an ***asymptotically optimal***, if for a given distribution of probabilities the ratio $AL(K)/AE(S)$ is close to 1, while the entropy is close to ∞.

Definition: A *code* K is a ***universal*** *one*, if there are $c_1, c_2 \in R$ so, that average length of code word $AL(K) \leq c_1 \times AE + c_2$.

Theorem: Universal *code* is ***asymptotically optimal***, if $c_1 = 1$.

Universal coding of integers

Definition: ***Fibonacci sequence of order m***

$F_n = F_{n-m} + F_{n-m+1} + \ldots + F_{n-1}$ for $n \geq 1$.

Example: F of order 2: $F_{-1} = 0,$, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8, \ldots$

Example: F of order 3: $F_{-2} = 0$, $F_{-1} = 0$, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 4$, $F_4 = 7$, $F_5 = 13, \ldots$

Example: F of order 4: $F_{-3} = 0$, $F_{-2} = 0$, $F_{-1} = 0$, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 4$, $F_4 = 8$, $F_5 = 15, \ldots$

Definition: ***Fibonacci representation*** $R(N) = \sum_{i=1}^{k} d_i F_i$, where $d_i \in \{0, 1\}$, $d_k = 1$

Theorem: Fibonacci representation is ambiguous, however there is such a one, that has at most $m - 1$ consecutive ones in a sequence d_i.

Fibonacci codes

Definition: ***Fibonacci code of order m*** $FK_m(N) = d_1 d_2 \dots d_k \underbrace{1 \dots 1}_{m-1 \text{ krát}}$,

where d_i are coefficients from previous sentence (ones end a word).
Example: $R(32) = 0*1+0*2+1*3+0*5+1*8+0*13+1*21$,
thus $F(32) = 00101011$.
Theorem: $FK(2)$ is a prefix, universal code with $c_1 = 2$, $c_2 = 3$, thus it isn't asymptotically optimal.

The universal coding of the integers II

- unary code $\alpha(N) = \underbrace{00\ldots0}_{N-1}1$.
- binary code $\beta(1) = 1, \beta(2N + j) = \beta(N)j,\ j = 0, 1$.
- β is not uniquely decodable (it isn't prefix code).
- ternary $\tau(N) = \beta(N)\#$.
- $\beta'(1) = \epsilon, \beta'(2N) = \beta'(N)0, \beta'(2N + 1) = \beta'(N)1, \tau'(N) = \beta'(N)\#$.
- γ: every bit $\beta'(N)$ is inserted between a pair from $\alpha(|\beta(N)|)$.
- example: $\gamma(6) = 0\overline{1}0\overline{0}1$
- $C_\gamma = \{\gamma(N) : N > 0\} = (0\{0, 1\})^*1$ is regular and therefore it's decodable by finite automaton.

The universal coding of the integers III

- $\gamma'(N) = \alpha(|\beta(N)|)\beta'(N)$ the same length (bit permutation $\gamma(N)$), but more readable
- $C_{\gamma'} = \{\gamma'(N) : N > 0\} = \{0^k 1\{0,1\}^k : k \geq 0\}$ is not regular and the decoder needs a counter
- $\delta(N) = \gamma(|\beta(N)|)\beta'(N)$
- example: $\delta(4) = \gamma(3)00 = 01100$
- decoder δ: $\delta(?) = 0011?$
- ω:

  $K := 0;$

  while $\lfloor \log_2(N) \rfloor > 0$ **do**

  begin $K := \beta(N)K;$

  $N := \lfloor \log_2(N) \rfloor$

  end.

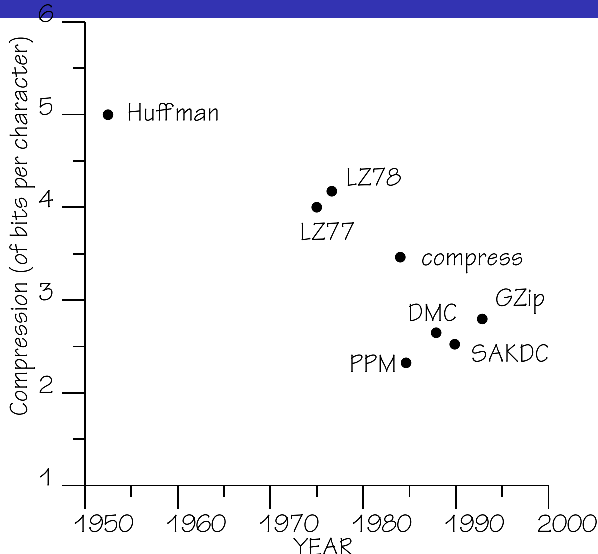
Data compression – introduction

- ☞ Information encoding for communication purposes.
- ☞ Despite tumultuous evolution of capacities for data storage, there is still a lack of space, or access to compressed data saves time. Redundancy $\longrightarrow$ a construction of a minimal redundant code.
- ☞ Data model:
  - structure – a set of units to compression + context of occurrences;
  - parameters – occurrence probability of particular units.
  - data model creation;
  - the actual encoding.

Data compression – *evolution*

- 1838 Morse, *code* *e* by frequency.
- 1949 Shannon, Fano, Weaver.
- 1952 Huffman; 5 bits per character.
- 1979 Ziv-Lempel; **`compress`** (Roden, Welsh, Bell, Knuth, Miller, Wegman, Fiala, *Green*, . . .); 4 bits per character.
- *eighties and nineties* PPM, DMC, **`gzip`** (zlib), SAKDC; 2–3 bits/character
- at the turn of millenium **`bzip2`**; 2 bits per character.
- . . . ?

Evolution of compression algorithms



Prediction and modeling

- redundancy (non-uniform probability of source unit occurrences)
- encoder, decoder, model
- statistical modeling (the model doesn't depend on concrete data)
- semiadaptive modeling (the model depends on data, 2 passes, necessity of model transfer)
- adaptive modeling (only one pass, the model is created dynamically by both encoder and decoder)

Prediction and modeling

- models of order 0 – probabilities of isolated source units (*e.g.* Morse, character *e*)
- models with a finite context – Markov models, models of order n (*e.g.* Bach), $P(a|x_1x_2\dots x_n)$
- models based on finite automata
  - synchronization string, nonsynchronization string
  - automaton with a finite context
  - suitable for regular languages, unsuitable for context-free languages, $P(a|q_i)$

Outline (Week twelwe)

- ☞ Huffman coding.
- ☞ Adaptive Huffman coding.
- ☞ Aritmetic coding.
- ☞ Dictionary methods.
- ☞ Signature methods.
- ☞ Similarity of documents.
- ☞ Compression using neural networks.

Statistical compression methods I

Character techniques

- null suppression – replacement of repetition ≥ 2 of character null, 255, special character S_c
- run-length encoding (RLE) – $S_c X C_c$ generalization to any repetitious character $\$ * * * * * *55 \longrightarrow \$S_c * 655$
- MNP Class 5 RLE – CXXX *DDDDDBBAAAA* $\longrightarrow$ *5DDDBB4AAA*
- half-byte packing, (EBCDIC, ASCII) SI, SO
- diatomic encoding; replacement of character pairs with one character.
- Byte Pair Encoding, BPE (Gage, 1994)
- pattern substitution
- Gilbert Held: Data & Image Compression

Statistical compression methods II

- Shannon-Fano, 1949, model of order 0,
- *code words of length* $\lfloor -\log_2 p_i \rfloor$ *or* $\lfloor -\log_2 p_i + 1 \rfloor$
- $AE \leq AL \leq AE + 1$.
- *code tree* (2,2,2,2,4,4,8).
- generally it is not optimal, *two passes of encoder through text*, static $\longrightarrow$x

Shannon-Fano coding

Input: a sequence of n source units $S[i]$, $1 \leq i \leq n$, in order of nondecreasing probabilities.
Output: n binary code words.

```
begin assign to all code words an empty string;
      SF-SPLIT(S)
end

procedure SF-SPLIT(S);
begin if |S| ≥ 2 then
      begin divide S to sequences S1 and S2 so, that both
              sequences have roughly the same total probability;
            add to all code words from S1 0;
            add to all code words from S2 1;
            SF-SPLIT(S1); SF-SPLIT(S2);
      end
end
```

Huffman coding

- Huffman coding, 1952.
- static and dynamic variants.
- $AEPL = \sum_{i=1}^{n} d[i]p[i]$.
- optimal code (not the only possible).
- $O(n)$ assuming ordination of source units.
- stable distribution $\longrightarrow$ preparation in advance.

Example: (2,2,2,2,4,4,8)

Huffman coding – sibling property

Definition: Binary tree have a ***sibling property*** if and only if

1. each node except the root has a sibling,
2. nodes can be arranged in order of nondecreasing sequence so, that each node (except the root) adjacent in the list with another node, is his sibling (the left sons are on the odd positions in the list and the right ones on even).

Huffman coding – properties of Huffman trees

Theorem: A binary prefix *code* is a Huffman *one* ⇔ it has the sibling property.

- ☞ $2n - 1$ nodes, max. $2n - 1$ possibilities,
- ☞ optimal binary prefix *code*, that is not the Huffman *one*.
- ☞ $AR(X) \leq p_n + 0{,}086$, p_n maximum probability of source unit.
- ☞ Huffman is a full *code*, (*poor error detection*).
- ☞ possible to extend to an ***affix code***, KWIC, left and right context, searching for $*X*$.

Adaptive Huffman coding

- FGK (Faller, Gallager, Knuth)
- suppression of the past by coefficient of forgetting, rounding, 1, r, r^2, r^n.
- linear time of coding and decoding regarding the word length.
- $AL_{HD} \leq 2AL_{HS}$.
- Vitter $AL_{HD} \leq AL_{HS} + 1$.
- implementation details, tree representation code tables.

Principle of arithmetic coding

- generalization of Huffman coding (probabilities of source units needn't be negative powers of two).
- order of source units; **Cumulative probability $cp_i = \sum_{j=1}^{i-1} p_j$** source units x_i with probability p_i.
- Advantages:
  - any proximity to entropy.
  - adaptability is possible.
  - speed.

Dictionary methods of data compression

Definition: ***Dictionary*** is a pair $D = (M, C)$, where M is a finite set of words of source language, C mapping M to the set of code words.
Definition: $L(m)$ denotes the length of code word $C(m)$ in bits, for $m \in M$.
Selection of source units:

- static (agreement on the dictionary in advance)
- semiadaptive (necessary two passes trough text)
- adaptive

Statical dictionary methods

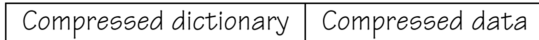
Source unit of the length n – n-grams
Most often bigrams ($n = 2$)

- n fixed
- n variable (by frequency of occurrence)
- adaptive

(50 % of an English text consits of about 150 most frequent words)
Disadvantages:

- they are unable to react to the probability distribution of compressed data
- pre-prepared dictionary

Semiadaptive dictionary methods



Advantages: *extensive date* (the dictionary is a small part of data – corpora; CQP).

Semiadaptive dictionary methods – dictionary creation procedure

1. The frequency of N-grams is determined for $N = 1, 2, \ldots$.
2. The dictionary is initialized by unigram insertion.
3. N-grams with the highest frequency are gradually added to the dictionary. During K-gram insertion frequencies decrease for it's components of $(K-1)$-grams, $(K-2)$-grams $\ldots$. If, by reducing of frequencies, a frequency of a component is greatly reduced, then it's excluded from the dictionary.

Outline (Week thirteen)

- Adaptive dictionary methods with dictionary restructuring.
- Syntactic methods.
- Checking of text correctness.
- Querying and TIS models.
- Vector model of documents
- Automatic text structuring.
- Document similarity.

Adaptive dictionary methods

LZ77 – siliding window methods
LZ78 – methods of increasing dictionary

a	b	c	b	a	b	b	a	a	b	a	c	b

encoded part (window, $N \leq 8192$)   not enc. part ($|B| \sim$10–20 b)

In the encoded part the longest prefix P of a string in not encoded part is searched. If such a string is found, then P is encoded using (I, J, A), where I is a distance of first character S from the border, J is a length of the string S and A is a first character behind the prefix P. The window is shifted by $J + 1$ characters right. If the substring S wasn't found, then a triple $(0, 0, A)$ is created, where A is a first character of not encoded part.

LZR (Rodeh)

$|M| = (N - B) \times B \times t$, t size of alphabet
$L(m) = \lceil \log_2 (N - B) \rceil + \lceil \log_2 B \rceil + \lceil \log_2 t \rceil$
Advantage: the search of the longest prefix [KMP]

- LZR uses a tree containing all the prefixes in the yet encoded part.
- The whole encoded yet encoded part is used as a dictionary.
- Because the i in (i, j, a) can be large, the Elias code for coding of the integers is used.

Disadvantage: a growth of the tree size without any limitation $\Rightarrow$ after exceeding of defined memory it's deleted and the construction starts from the beginning.

LZSS (Bell, Storer, Szymanski)

The *code* is a sequence of pointers and characters. The pointer (i, j) needs a memory as p characters $\Rightarrow$ a pointer only, when it pays off, but there is a bit needed to distinguish a character from a pointer. The count of dictionary items is $|M| = t + (N - B) \times (B - p)$ (considering only substrings longer than p). The bit count to encode is

- $L(m) = 1 + \lceil \log_2 t \rceil$ for $m \in T$
- $L(m) = 1 + \lceil \log_2 N \rceil + \lceil \log_2 (B - p) \rceil$ otherways.

(The length d of substring can be represented as $B - p$).

LZB (Bell), LZH (Brent)

A pointer (i, j) (analogy to LZSS)
If

- the window is not full (at the beginning) and
- the compressed text is shorter than N,

the usage of $\log_2 N$ bytes for encoding of i is a waste. LZB uses phasing for binary coding. – prefix code with increasing count of bits for increasing values of numbers. Elias code γ.
LZSS, where for pointer encoding the Huffman coding is used (i.e. by distribution of their probabilities $\Rightarrow$ 2 throughpasses)

Methods with increasing dictionary

The main idea: the dictionary contains phrases. A new phrase so, that an already existing phrase is extended by a symbol. A phrase is encoded by an index of the prefix and by the added symbol.

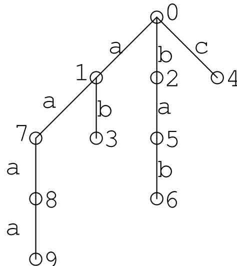
LZ78 – example

Input	a	b	ab	c	ba
Index	1	2	3	4	5
Output	(0,a)	(0,b)	(1,b)	(0,c)	(2,a)

…

…

Input	bab	aa	aaa	aaaa
Index	6	7	8	9
Output	(5,b)	(1,a)	(7,a)	(8,a)



LZFG (Fiala, Green)

A dictionary is stored in a tree structure, *edges are labeled with strings of characters*. These strings are in the window and *each node of the tree contains a pointer to the window and identifying symbols on the path from the root to the node.*

LZW (Welch), LZC

The output indexes are only, or

- the dictionary is initiated by items for all input symbols
- the last symbol of each phrase is the first symbol of the following phrase.

Input	a	b	a	b	c	b	a	b	a	b	a	a	a	a	a
Index		4	5		6	7		8			9	10			
Output	1	2		4	3		5			8	1		10	11	

Overflow ⇒ next phrase is not transmitted and coding continues statically.
it's a LZW +

- Pointers are encoded with prolonging length.
- Once the compression ratio will decrease, dictionary will be deleted and it starts from the beginning.

LZT, LZMW, LZJ

As LZC, but when a dictionary overflows, phrases, that were least used in the recent past, are excluded from the dictionary. It uses phrasing for binary coding of phrase indexes.
As LZT, but a new phrase isn't created by one character addition to the previous phrase, but the new phrase is constructed by concatenation of two last encoded ones.
Another principle of dictionary construction.

- At the beginning only the single symbols are inserted.
- Dictionary is stored in a tree and contains all the substrings processed by string of the length up to h.
- Full dictionary ⇒
  - statical procedure,
  - omitting of nodes with low usage frequency.

Dictionary methods with dictionary restructuring

- Ongoing organization of source units ⟶ shorter strings of the code.
- Variants of heuristics (count of occurrences, moving to the beginning (BSTW), change the previous, transfer of X forward).
- BSTW (advantage: high locality of occurrences of a small number of source units.
- Example: I'm not going to the forest, ..., $1^n 2^n k^n$.
- Generalization: ***recency coefficient***, ***Interval coding***.

Interval coding

Representation of the word by total sum of words from the last occurrence.
The dictionary contains words $a_1, a_2, \ldots, a_n$, input sequence contains $x_1, x2, \ldots, x_m$. The value LAST(a_i) containing the interval form last occurrence is initialized to zero.

for $t := 1$ *to* m **do**
begin $\{x_t = a_i\}$
  if $\mathrm{LAST}(x_t = 0)$ **then** $y(t) = t + i - 1$
    else $y(t) = t - \mathrm{LAST}(x_t)$;
  $\mathrm{LAST}(x_t) := t$
end .

Sequence $y_1, y_2, \ldots, y_m$ is an output of encoder and can be encoded by one code of variable length.

Syntactical methods

- the grammar of the message language is known.
- left partition of derivational tree of string.
- global numbering of rules.
- local numbering of rules.

- Decision-making states of LR analyzer are encoded.

Context modeling

- fixed context – model of order N.
- combined approach – contexts of various length.
- $p(x) = \sum_{n=0}^{m} w_n p_n(x)$.
- w_n fixed, variable.
- time and memory consuming.
- assignment of probability to the new source unit: $e = \frac{1}{C_n+1}$.
- automata with a finite context.
- dynamic Markov modeling.

Checking the correctness of the text

- Checking of text using frequency dictionary.
- Checking of text using a double frequency dictionary.
- Interactive control of text (ispell).
- Checking of text based on regularity of words, ***weirdness coefficient***.

Weirdness coefficient

Weirdness coefficient of trigram xyz

$KPT = [\log(f(xy) - 1) + \log(f(yz) - 1)]/2 - \log(f(xyz) - 1)$, where $f(xy)$ resp. $f(xyz)$ are relative frequencies of bigram resp. trigram, $\log(0)$ is defined as -10.

Weirdness coefficient of word $KPS = \sqrt{\sum_{i=1}^{n}(KPT_i - SKPT^2)}$, where

KPT_i is a weirdness coefficient of i-th trigram $SKPT$ is a mean rate of weirdness coefficients of all trigrams contained in the word.

Outline (Week fourteen)

- Querying and TIS models.
- Boolean model of documents.
- Vector model of documents.
- TIS Architecture.
- Signature methods.
- Similarity of documents.
- Vector model of documents (completion).
- Extended boolean model.
- Probability model.
- Model of document clusters.
- TIS Architecture.
- Automatic text structuring.
- Documents similarity.
- Lexicon storage.

Querying and TIS models

Different methods of hierarchization and document storage ⟶ different possibilities and efficiency of querying.

- Boolean model, SQL.
- Vector model.
- Extended boolean types.
- Probability model.
- Model of document clusters.

Blair's query tuning

The search lies in reducing of uncertainty of a question.

1. We find a document with high relevance.
2. We start to query with it's key words.
3. We remove descriptors, or replace them with disjunctions.

Infomap – attempt to semantic querying

System **http://infomap.stanford.edu** – for working with searched meaning/concept (as opposed to mere strings of characters).

Right query formulation is the half of the answer. The search lies in determination of semantically closest terms.

Boolean model

- Fifties: representation of documents using sets of terms and querying based on evaluation of boolean expressions.
- Query expression: inductively from primitives:
  - term
  - attribute_name = attribute_value (comparison)
  - function_name(term) (application of function)

  and also using parentheses and logical conjunctions X and Y, X or Y, X xor Y, not Y.
- disjunctive normal form, conjunctive normal form
- proximity operators
- regular expressions
- thesaurus usage

Languages for searching – SQL

- boolean operators ***and***, ***or***, ***xor***, ***not***.
- positional operators ***adj***, ***(n) words***, ***with***, ***same***, ***syn***.
- SQL extension: operations/queries with use of thesaurus

BT(A)	Broader term
NT(A)	Narrower term
PT(A)	Preferred term
SYN(A)	Synonyms of the term A
RT(A)	Related term
TT(A)	Top term

Querying – SQL examples

```
ORACLE SQL*TEXTRETRIEVAL
SELECT specification_of_items
FROM specification_of_tables
WHERE item
CONTAINS textov_expression
```

Example:

```
SELECT TITLE
FROM BOOK
WHERE ABSTRACT
CONTAINS 'TEXT' AND RT(RETRIEVAL)
'string' 'string'*  *'string' 'st?ing'
'str%ing' 'stringa' (m,n) 'stringb'
'multiword phrases' BT('string',n)
BT('string',*) NT('string',n)
```

Querying – SQL examples

Example:

```
SELECT NAME
FROM EMPLOYEE
WHERE EDUCATION
CONTAINS  RT(UNIVERSITA)
AND LANGUAGES
CONTAINS 'ENGLISH' AND 'GERMAN'
AND PUBLICATIONS
CONTAINS 'BOOK' OR NT('BOOK',*)
```

Stiles technique/ association factor

$$asoc(Q_A, Q_B) = \log_{10} \frac{(fN - AB - N/2)^2 N}{AB(N-A)(N-B)}$$

A – number of documents „hit" by the query Q_A
B – number of documents „hit" by the query Q_B (its relevance we count)
f – number of documents „hit" by both the queries
N – total sum of documents in TIS
cutoff (relevant/ irrelevant)
clustering/nesting 1. generation, 2. generation, ...

Vector model

Vector model of documents: Let $a_1, \ldots, a_n$ be terms, $D_1, \ldots, D_m$ documents, and ***relevance matrix*** $\boldsymbol{W = (w_{ij})}$ of type m, n,

$$w_{ij} \in \langle 0, 1 \rangle \begin{cases} 0 & \text{is irrelevant} \\ 1 & \text{is relevant} \end{cases}$$

Query $\boldsymbol{Q = (q_1, \ldots, q_n)}$

- $S(Q, D_i) = \sum_i q_i w_{ij}$ ***similarity coefficient***
- $head(sort(S(Q, D_i)))$ answer

Vector model: pros & cons

CONS: doesn't take into account ?"and"? ?"or"?

PROS: possible improvement:

- normalization of weights
  - **Term frequency TF**
  - **Inverted document frequency** $IDF \equiv \log_2 \frac{m}{k}$
  - Distinction of terms
- normalization of weights for document: $\frac{TD}{\sqrt{\sum_j TD_j^3}}$
- normalization of weights for query: $\left(\frac{1}{2} \times \frac{\frac{1}{2}TF}{\max TF_i}\right) \times \log_2 \frac{m}{k}$

[POK, pages 85–113].

Automatic structuring of texts

- Interrelations between documents in TIS.
- Encyclopedia (OSN, Funk and Wagnalls New Encyclopedia).
- [SBA]
  `http://columbus.cs.nott.ac.uk/compsci/epo/epodd/ep056gs`
- *Google*/CiteSeer: „automatic structuring of text files“

Similarity of documents

- ☞ Most often cosine measure – advantages.
- ☞ Detailed overview of similarity functions see chapter 5.7 from [KOR] (similarity).

Lexicon storage

[MeM] Mehryar Mohri: On Some Applications of Finite-State Automata Theory to Natural Language Processing, *Natural Language Engineering*, 2(1):61–80, 1996.
`http://www.research.att.com/~mohri/cl1.ps.gz`