Modern Information Retrieval

Chapter 6

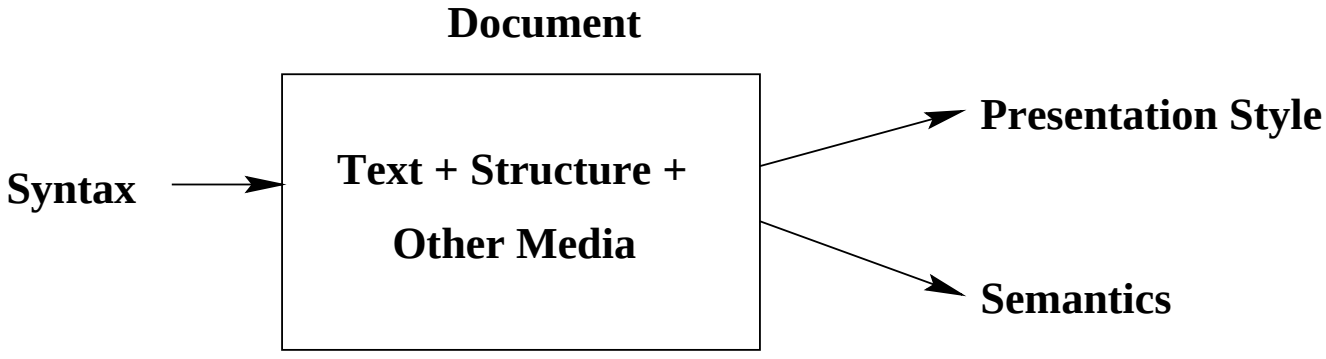
Documents: Languages & Properties

Metadata
Document Formats
Markup Languages
Text Properties
Document Preprocessing
Organizing Documents
Text Compression

Introduction

- The **document**
  - denotes a single unit of information
  - has a **syntax and structure**
  - has a **semantics**, specified by the author
  - may have a **presentation style**
    - given by its syntax and structure
    - related to a specific application
    - specifies how to display or print document

Introduction



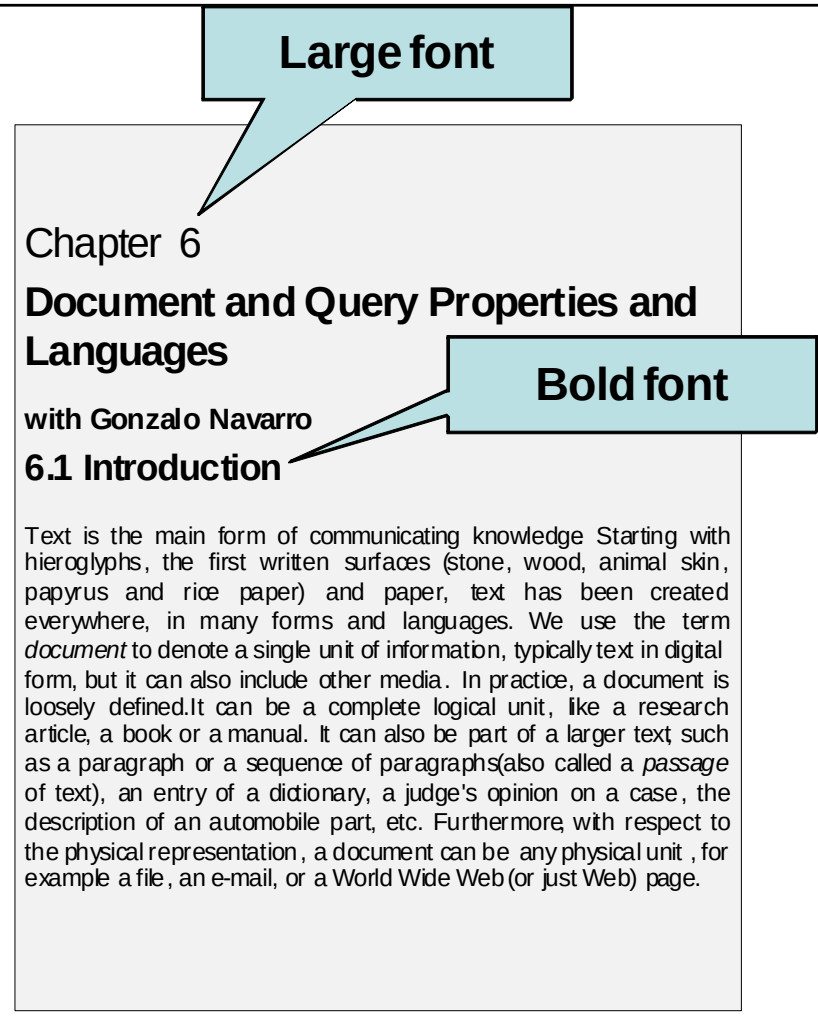
- The **document syntax**
  - expresses structure, presentation style, semantics
  - one or more of elements might be implicit or given together
  - structural element (e.g., a section) can have fixed formatting style

Introduction

- The **document syntax** can be
  - **implicit** in its content
  - expressed in a **simple declarative language**
  - expressed in a **programming language**
    - the language syntax might be proprietary and specific
    - open and generic languages are more flexible
- Text can also be written in **natural language**
  - hard to process using a computer
- Current trend: use document languages that
  - provide information on structure, format, and semantics
  - are readable by humans and computers

Introduction

- **Document style**
- defines how a document is visualized or printed
- can be embedded in the document: TeX and RTF
- can be complemented by macros: LaTeX



Introduction

- **Queries in search engines**
  - can be considered as short pieces of text
  - differ from normal text
  - understanding them is very important
  - semantics often ambiguous due to polysemy
  - not simple to infer user intent behind a query

Metadata

Metadata

- **Metadata** is information on the organization of the data, the various data domains, and their relationship
  - metadata is **data about the data**
    - in a database, names of relations and attributes constitute metadata
  - metadata is associated with most documents and text collections

Descriptive Metadata

- Common forms of metadata for documents
  - author of the text
  - date of publication
  - source of the publication
  - document length
- Dublin Core Metadata Element Set proposes 15 fields to describe a document
- Marchionini refers to this type of information as **Descriptive Metadata**
  - Descriptive metadata are external to the meaning of the document and pertain more to how it was created

Semantic Metadata

- **Semantic Metadata**
  - characterizes the subject matter within the document contents
  - is associated with a wide number of documents
  - its availability is increasing
- An important metadata format is **MARC (Machine Readable Cataloging Record)**
  - most used format for library records
  - includes fields for distinct attributes of a bibliographic entry such as title, author, publication venue
  - In the U.S.A., a particular version of MARC is used: **USMARC**

Metadata in Web Documents

- The increase in Web data has led to many initiatives to add metadata information to Web pages for various purposes such as
  - cataloging and content rating
  - intellectual property rights and digital signatures
  - applications to electronic commerce
- **RDF (Resource Description Framework)**
  - new standard for Web metadata
  - allows describing Web resources to facilitate automated processing

Metadata in Web Documents

- RDF does not assume any particular application or semantic domain
- It consists of a description of nodes and attached **attribute/value pairs**
  - Nodes can be any Web resource, that is, any **Uniform Resource Identifier (URI)** including **Uniform Resource Locators (URLs)**
  - Attributes are properties of nodes and their values are text strings or other nodes (Web resources or metadata instances)
- More details about RDF ahead

Document Formats

Text

- With the advent of the computer, it became necessary to represent code characters in binary digits, which is done through **coding schemes**
  - EBCDIC (7 bits), ASCII (8 bits) and UNICODE (16 bits)
  - All these coding schemes are based on characters
- An IR system should be able to retrieve information from many text formats (doc, pdf, html, txt)
- IR systems have filters to handle most popular documents
  - But, good filters might not be possible with proprietary formats

Text

- Other text formats
  - **Rich Text Format (RTF)**: for document interchange
  - **Portable Document Format (PDF)**: for printing and displaying
  - **Postscript**: for printing and displaying
- Other interchange formats are used to encode electronic mail
  - **Multipurpose Internet Mail Exchange (MIME)**: for encoding email
  - **Compress** (Unix), **ARJ** (PCs): for compressing text
  - **ZIP** (Unix) (`gzip` in Unix and `Winzip` in Windows): for compressing text

Multimedia

- Multimedia usually stands for applications that handle different types of digital data
  - Most common types of media: text, sound, images, and video
  - Different types of formats are necessary for storing each media
  - Most formats for multimedia can only be processed by a computer
- Additional information is given in the Multimedia Retrieval section

Image Formats

- The simplest image formats are direct representations of a bit-mapped display such as XBM, BMP or PCX
- Images of these formats have a lot of redundancy and can be compressed efficiently
  - Example of format that incorporates compression:
    Compuserve's Graphic Interchange Format (GIF)

Image Formats

- To improve compression ratios, lossy compression was developed
  - uncompressing a compressed image does not yield exactly the original image
- This is done by the **Joint Photographic Experts Group (JPEG)** format
  - JPEG tries to eliminate parts of the image that have less impact in the human eye
  - This format is parametric, in the sense that the loss can be tuned

Image Formats

- Another common image format is the **Tagged Image File Format** (TIFF)
  - exchange of documents between different applications and computers
  - TIFF provides for metadata, compression, and varying number of colors
- Yet another format is **Truevision Targa image file (TGA)**, which is associated with video game boards

Image Formats

- In 1996, a new image format was proposed for use in the Internet: Portable Network Graphics (PNG)
  - Standard de facto for images in the Web
- Further, various other image formats are associated with particular applications ranging from fax to fingerprints

Audio

- Audio must be digitalized to be stored properly
- Most common formats for audio: **AU**, **MIDI** and **WAVE**
  - MIDI: standard format to interchange music between electronic instruments and computers
- For audio libraries other formats are used such as **RealAudio** or **CD formats**

Movies

- Main format for animations is **Moving Pictures Expert Group (MPEG)**:
  - works by coding the changes in consecutive frames
  - profits from the temporal image redundancy that any video has
  - includes the audio signal associated with the video
  - specific cases for audio (MP3), video (MP4), etc.
- Other video formats are **AVI**, **FLI** and **QuickTime**
  - AVI may include compression (CinePac)
  - QuickTime, developed by Apple, also includes compression

Graphics and Virtual Reality

- There are many formats for three dimensional graphics:
  - **Computer Graphics Metafile (CGM)** and **Virtual Reality Modeling Language (VRML)**
- VRML, intended to be a universal interchange format for 3D graphics and multimedia, may be used in a variety of application areas such as
  - engineering and scientific visualization
  - multimedia presentations
  - entertainment and educational titles
  - web pages and shared virtual worlds
- VRML has become the *de facto* standard Modeling Language for the Web

Markup Languages

Markup Languages

- Markup is defined as extra syntax used to describe formatting actions, structure information, text semantics, attributes
- Examples of Markup Languages
  - **SGML**: Standard Generalized Markup Language
  - **XML**: eXtensible Markup Language
  - **HTML**: Hyper Text Markup Language

Markup Languages

SGML

SGML

- SGML (ISO 8879) stands for **Standard Generalized Markup Language**, i.e., a meta-language for tagging text
  - it provides rules for defining a markup language based on tags
  - includes a description of the document structure called **document type definition**
- An SGML document is defined by:
  - a document type definition; and
  - the text itself marked with tags which describe the structure

SGML

- The **document type definition** is used to
  - describe and name the pieces that a document is composed of
  - define how those pieces relate to each other
  - part of the definition can be specified by an **SGML document type declaration (DTD)**
- Other parts, such as the semantics of elements and attributes, or application conventions, cannot be expressed formally in SGML
  - Comments can be used, however, to express them informally
  - More complete information is usually present in separate documentation

SGML

- Tags are denoted by angle brackets
  - Tags are used to identify the **beginning** and **ending** of an element
  - Ending tags include a slash before the tag name
  - **Attributes** are specified inside the beginning tag

SGML

- Example of a SGML DTD for electronic messages

```
<!ELEMENT e-mail                - - (prolog, contents) >
<!ELEMENT prolog                - - (sender, address+, subject?, Cc*) >
<!ELEMENT (sender | address | subject | Cc) - O (#PCDATA) >
<!ELEMENT contents              - - (par | image | audio)+ >
<!ELEMENT par                   - O (ref | #PCDATA)+ >
<!ELEMENT ref                   - O EMPTY >
<!ELEMENT (image | audio)       - -  (#NDATA) >

<!ATTLIST e-mail
          id            ID                      #REQUIRED
          date_sent     DATE                    #REQUIRED
          status        (secret | public )      public >
<!ATTLIST ref
          id            IDREF                   #REQUIRED >
<!ATTLIST (image | audio )
          id            ID                      #REQUIRED >
```

SGML

- Example of use of previous DTD

```
<!DOCTYPE e-mail SYSTEM "e-mail.dtd">
<e-mail id=94108rby date_sent=02101998>
  <prolog>
  <sender> Pablo Neruda </sender>
  <address> Federico Garcia Lorca </address>
  <address> Ernest Hemingway </address>
  <subject> Pictures of my house in Isla Negra
  <Cc> Gabriel Garcia Marquez </Cc>
  </prolog>
  <contents>
      <par>
         As promised in my previous letter...
      </par>
  </contents>
</e-mail>
```

SGML

- Document description does not specify how a document is printed
  - Output specifications are often added to SGML documents, such as:
    - **DSSSL**: Document Style Semantic Specification Language
    - **FOSI**: Formatted Output Specification Instance
  - These standards define mechanisms for associating style information with SGML document instances
  - They allow defining that the data identified by a tag should be typeset in some particular font

SGML

- One important use of SGML is in the **Text Encoding Initiative (TEI)**
  - includes several USA associations related to the humanities and linguistics
  - provides several document formats through SGML DTDs
  - main goal is to generate guidelines for the preparation and interchange of electronic texts for scholarly research, as well as the industry
  - one of the most used formats is **TEI Lite**

Markup Languages

HTML

HTML

- HTML stands for **HyperText Markup Language** and is an instance of SGML
- It was created in 1992 and has evolved during the last years, being 4.0 the latest version
- HTML5 is under development
- Most documents on the Web are stored and transmitted in HTML
- Although there is an HTML DTD, most HTML instances do not explicitly make reference to the DTD
- The HTML tags follow all the SGML conventions and also include formatting directives

HTML

- HTML documents can have other media embedded within them, such as images or audios
- HTML also has fields for **metadata**, which can be used for different applications and purposes
- If we also add programs (for example, using Javascript) inside a page some people call it **dynamic HTML**

HTML

- Example of an HTML document

```
<html><head>
<title>HTML Example</title>
<meta name=rby content="Just an example">
</head>
<body>
<h1>HTML Example</h1>
<p><hr><p>HTML has many <i>tags</i>, among them:
<ul>
<li> links to other <a href=example.html>pages</a> (a from anchor),
<li> paragraphs (p), headings (h1, h2, etc), font types (b, i),
<li> horizontal rules (hr), indented lists and items (ul, li),
<li> images (img), tables, forms, etc.
</ul>
<p><hr><p><img align=left src="at_work.gif">
This page is <b>always</b> under construction.
</body></html>
```

HTML

- How the HTML document is seen on a browser

HTML Example

HTML has many *tags*, among them:

- links to other pages (a from anchor),
- paragraphs (p), headings (h1, h2, etc), font types (b, i),
- horizontal rules (hr), indented lists and items (ul, li),
- images (img), tables, forms, etc.

This page is **always** under construction.

HTML

- Because HTML does not fix a presentation style, the **Cascade Style Sheets** (CSS) were introduced in 1997
  - powerful and manageable way for authors to improve the aesthetics of HTML pages
  - separate information about presentation from document content
  - support (for CSS) in current browsers is still modest

HTML

- The evolution of HTML implies support for backward compatibility and also for forward compatibility
- HTML 4.0 has been specified in three flavors: strict, transitional, and frameset
  - **Strict HTML** only worries about non-presentational markup, leaving all the displaying information to CSS
  - **Transitional HTML** uses all the presentational features for pages that should be read for old browsers that do not understand CSS
  - **Frameset HTML** is used when you want to partition the browser window in two or more frames
- HTML 4.0 includes support for style sheets, internationalization, frames, richer tables and forms, and accessibility options for people with disabilities

HTML

- Typical HTML applications use a fixed small set of tags
  - makes the language specification much easier to build applications
  - comes at the cost of severely limiting HTML in several important aspects
  - In particular, HTML does not
    - allow users to specify their own tags
    - support the specification of nested structures needed to represent database schemas
    - support the kind of language specification that allows consuming applications to check data for structural validity on importation

Markup Languages

XML

XML

- XML , the **eXtensible Markup Language**
  - is a simplified subset of SGML
  - is not a markup language, as HTML, but a meta-language, as SGML
  - allows to have human-readable semantic markup, which is also machine-readable
  - makes it easier to develop and deploy new specific markup languages

XML

- XML does not have many of the restrictions of HTML
- On the other hand, imposes a more rigid syntax on the markup:
  - In XML, ending tags cannot be omitted
  - XML also distinguishes upper and lower case
  - All attribute values must be between quotes
  - Parsing XML without a DTD is easier
    - The tags can be obtained while the parsing is done

XML

- XML allows any user to define new tags
- **Extensible Style sheet Language (XSL)**
  - the XML counterpart of Cascading Style Sheets (CSS)
  - syntax defined based on XML
  - designed to transform and style highly-structured, data-rich documents written in XML
    - For example, with XSL it would be possible to automatically extract a table of contents from a document

XML

- **Extensible Linking Language (XLL)**
  - Another extension to XML, defined using XML
  - defines different types of links (external and internal)
- Recent uses of XML include:
  - Mathematical Markup Language (MathML)
  - Synchronized Multimedia Integration Language (SMIL)
  - Resource Description Format
- Next generation of HTML should be based in a suite of XML tag sets

Markup Languages

RDF: Resource Description Framework

Resource Description Framework

- RDF (Resource Description Framework)
  - family of specifications originally designed as a metadata model
  - has become a general method for the conceptual description or modeling of information
  - the **de facto** standard language of the Semantic Web
  - similar to conceptual modeling approaches such as Entity-Relationship or Class diagrams
  - more naturally suited to represent certain kinds of knowledge than other traditional models

Resource Description Framework

- The RDF Schema (RDFS) is an extensible knowledge representation language
  - It provides basic elements for the description of ontologies, intended to structure RDF resources
- Many RDFS components are included in the more expressive language Web Ontology Language (OWL)
  - OWL is a family of knowledge representation languages for authoring ontologies, also endorsed by the W3C
- Ontologies defined with OWL are most commonly serialized using RDF/XML syntax

Markup Languages

HyTime

HyTime

- HyTime: **Hypermedia/Time-based Structuring Language**
  - an SGML architecture that specifies the generic hypermedia structure of documents
  - its hypermedia concepts include
    - complex locating of document objects
    - relationships (hyperlinks) between document objects
    - numeric, measured associations between document objects

HyTime

- The HyTime architecture has three parts:
  - the base **linking and addressing architecture**
  - the **scheduling architecture** (derived from the base architecture)
  - the **rendition architecture** (which is an application of the scheduling architecture)
- HyTime does not directly specify graphical interfaces, user navigation or user interaction
- These aspects of document processing are rendered from the HyTime constructs in a similar manner as style sheets in SGML documents

Text Properties

Information Theory

Information Theory

- It is difficult to formally capture **how much information** is there in a given text
- However, the distribution of symbols is related to it
  - For example, a text where one symbol appears almost all the time does not convey much information
  - Information theory defines a special concept, **entropy**, to capture information content

Entropy

- The entropy of a whole text $T = t_1\ t_2 \ldots t_n$ is given by

$$E = \frac{1}{n} \sum_{i=1}^{n} \log_2 \frac{1}{p_i}$$

  where p_i is the probability assigned by the model to symbol t_i

- It is important to note that E is a property of the model and not only of the text

Entropy

- If the model always assign probability p_i to the alphabet symbol s_i, the entropy can be rewritten as

$$E = \sum_{s_i \in \Sigma} p_i \log_2 \frac{1}{p_i}$$

or

$$\boxed{E = -\sum_{i=1}^{\sigma} p_i \log_2 p_i}$$

where Σ is the alphabet of the text and $\sigma = |\Sigma|$

- Entropy is also a limit on how much a text can be compressed

Text Properties

Modeling Natural Language

Modeling Natural Language

- We can divide the symbols of a text in two disjoint subsets:
  - symbols that separate words; and
  - symbols that belong to words
- It is well known that symbols are not uniformly distributed in a text
  - For instance, in English, the vowels are usually more frequent than most consonants

Modeling Natural Language

- A simple model to generate text is the **Binomial model**
- However, the probability of a symbol depends on previous symbols
  - For example, in English, a letter **f** cannot appear after a letter **c**
- We can use a finite-context or **Markovian model** to reflect this dependency
- More complex models include finite-state models, and grammar models
- However, finding the right grammar for natural language is still a difficult open problem

Modeling Natural Language

- The second issue is **how the different words are distributed** inside each document
- An approximate model is the **Zipf's Law**
  - This law states that the frequency f_i of the i-th most frequent word is given by

$$f_i = \frac{f_1}{i^\alpha}$$

where f_1 is the frequency of the most frequent word and α is a text dependent parameter

Modeling Natural Language

- For a text of n words with a vocabulary of V words, we have

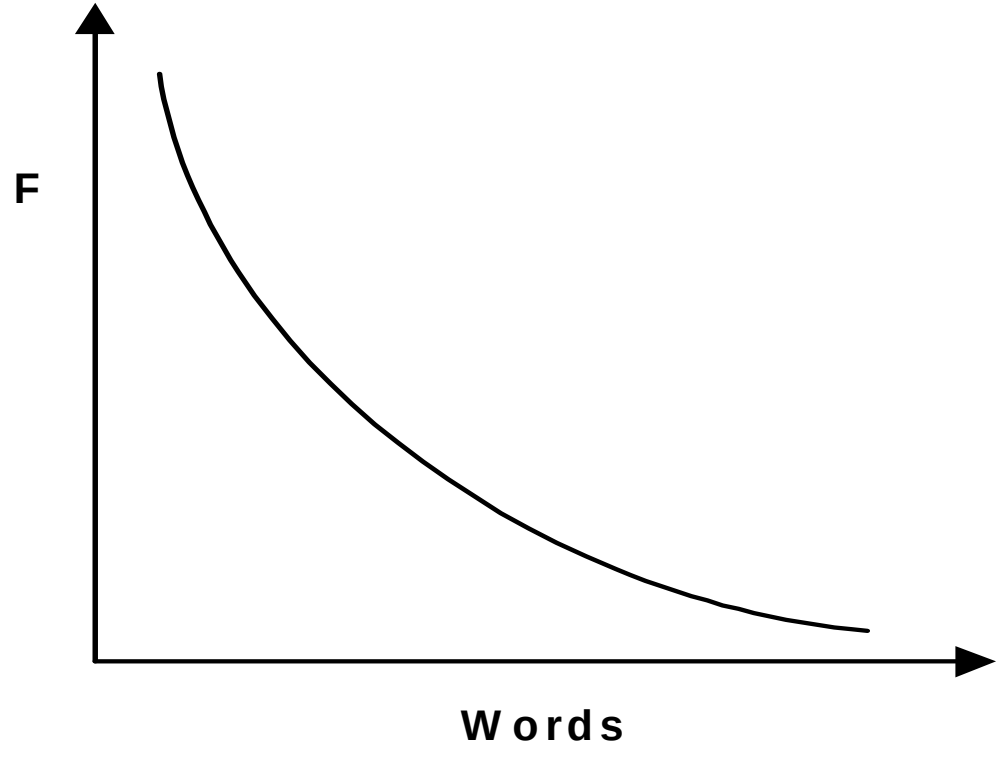
$$n = \sum_{i=1}^{V} \frac{1}{i^{\alpha}} f_1 = f_1 \times \left[\sum_{i=1}^{V} \frac{1}{i^{\alpha}} \right]$$

- The factor enclosed in brackets depends only on the text parameters α and V
- It is called the harmonic number $H_V(\alpha)$ of order α of V, that is

$$H_V(\alpha) = \sum_{i=1}^{V} \frac{1}{i^{\alpha}} \quad \text{and then} \quad \boxed{f_1 = \frac{n}{H_V(\alpha)}}$$

Modeling Natural Language

- Figure below illustrates the distribution of frequencies of the terms in a text
  - words arranged in decreasing order of their frequencies



Modeling Natural Language

- Since the distribution of words is very skewed, words that are too frequent, called stopwords, can be disregarded
- A **stopword** is a word which does not carry meaning in natural language
  - Examples of stopwords in english: *a*, *the*, *by*, *and*
  - Fortunately, the most frequent words are stopwords
    - Therefore, half of the words appearing in a text do not need to be considered

Modeling Natural Language

- A third issue is the **distribution of words** in the documents of a collection
- Simple model: consider that each word appears the same number of times in every document (not true)
- Better model: use a negative binomial distribution
  - Fraction of documents containing a word k times is

$$F(k) = \binom{\alpha + k - 1}{k} p^k (1 + p)^{-\alpha - k}$$

where p and α are parameters that depend on the word and the document collection

Modeling Natural Language

- The fourth issue is the **number of distinct words** in a document (the **document vocabulary**)
- To predict the growth of the vocabulary size in natural language text, we use the so called **Heaps' Law**
  - the vocabulary of a text of n words is of size

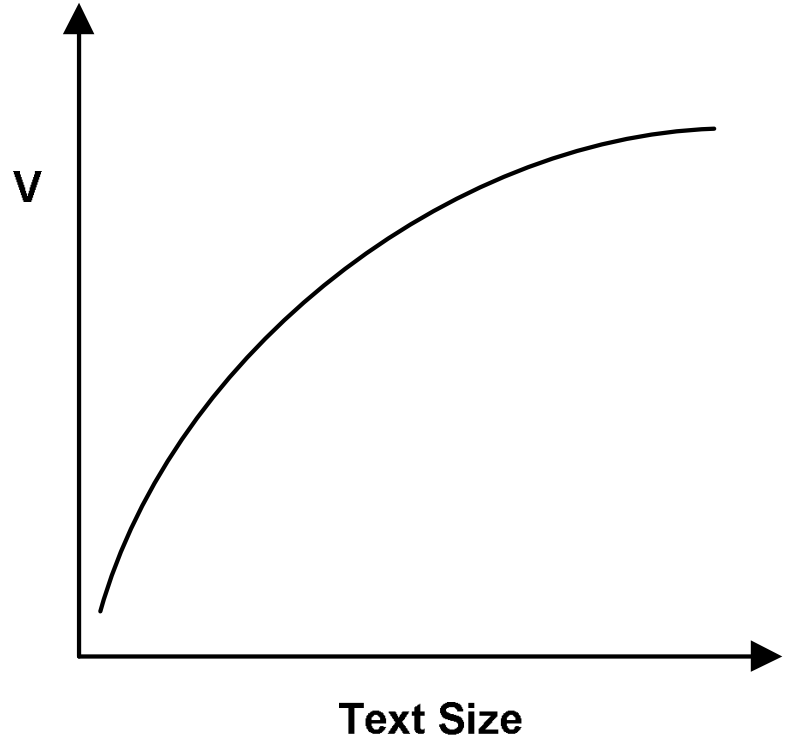
$$\boxed{V = Kn^{\beta}}$$

or $O(n^{\beta})$, where K and β depend on the text. Usually,

- $10 \leq K \leq 100$
- $0 < \beta < 1$
- in the TREC-2 collection, commonly $0.4 \leq \beta \leq 0.6$

Modeling Natural Language

- The figure below illustrates that vocabulary size grows sub-linearly with text size



Modeling Natural Language

- Notice that the set of different words of a language is fixed by a constant
- However, the limit is so high that it is common assume that the size of the vocabulary is $O(n^{\beta})$ instead of $O(1)$
- Many authors argue that the number keeps growing anyway because of typing and spelling errors
- Heap's law also applies to collections of documents
  - as the total text size grows, the predictions of the model become more accurate
  - the model is also valid for the World Wide Web

Modeling Natural Language

- A last issue is the **average length of words**
  - It relates the text size in words with the text size in bytes
  - In sub-collections of TREC-2, average word length is very close to 5 letters
  - If we remove the stopwords, the average length of a word increases to a number between 6 and 7 letters

Text Properties

Text Similarity

Text Similarity

- Similarity is measured by a **distance function**
  - for strings of the same length, distance between them is the number of positions with different characters
  - for instance, the distance is 0 if they are equal
  - this is called the **Hamming distance**
- A distance function should also be **symmetric**
  - In this case, the order of the arguments does not matter
- A distance function should also satisfy the triangle inequality:

$$distance(a, c) \leq distance(a, b) + distance(b, c)$$

Text Similarity

- **Edit or Levenshtein distance**
  - important distance function over strings
  - it is the minimal number of char insertions, deletions, and substitutions needed to make two strings equal
  - edit distance between `color` and `colour` is 1
  - edit distance between `survey` and `surgery` is 2
- Extensions to this concept
  - different weights for each operation
  - adding transpositions

Text Similarity

- **Longest common subsequence (LCS)**
  - all non-common characters of two (or more) strings
  - remaining sequence of characters is the LCS of both strings
  - LCS of `survey` and `surgery` is `surey`

Text Similarity

- Similarity can be extended to documents
- Consider lines as single symbols and compute the longest common sequence of lines between two files
  - Measure used by the `diff` command in Unix
  - Problems with this approach
    - very time consuming
    - does not consider lines that are similar
    - but, this can be fixed by taking a weighted edit distance between lines

Text Similarity

- **Resemblance measure**
  - If $W(d_j)$ is the set of all distinct words in document d_j, then the resemblance function between two documents d_i and d_j is defined as

$$R(d_i, d_j) = \frac{|W(d_i) \cap W(d_j)|}{|W(d_i) \cup W(d_j)|}$$

    where $0 \leq R(d_i, d_j) \leq 1$
  - Notice that this is a more efficient document similarity measure
- This resemblance measure can be easily transformed in a distance function $D(d_i, d_j)$

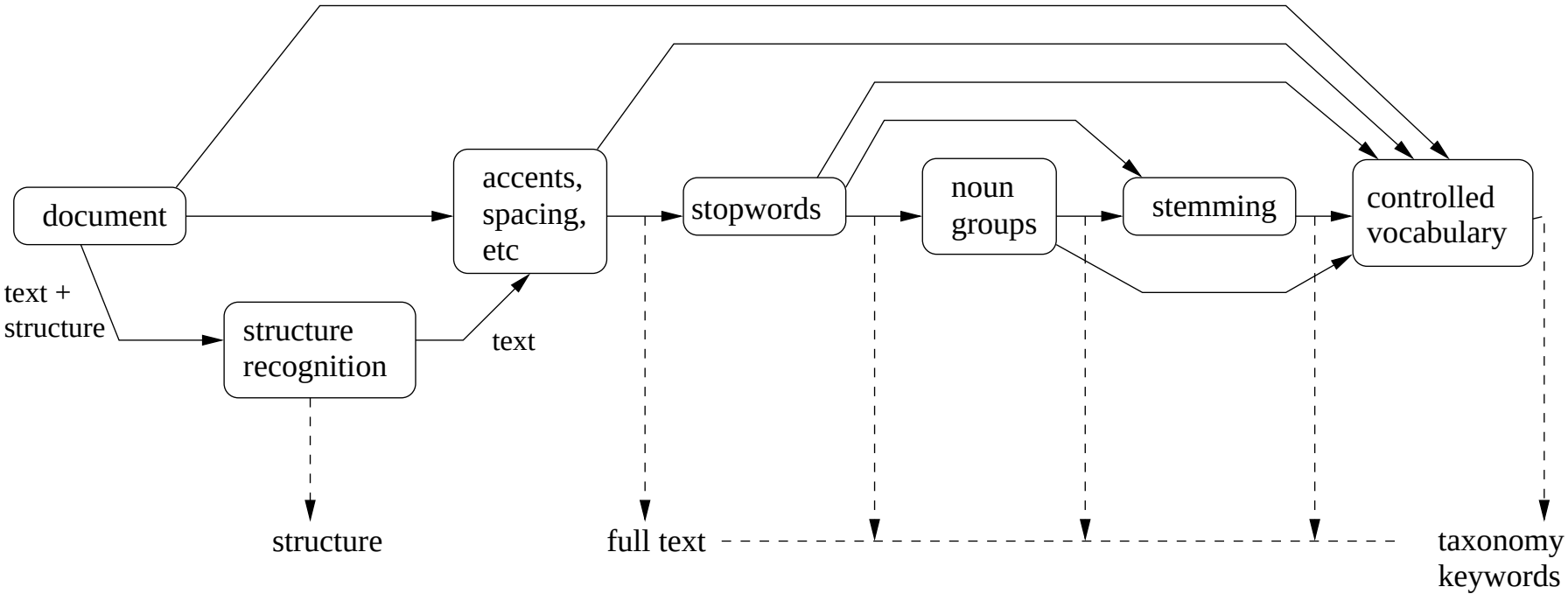
$$D(d_i, d_j) = 1 - R(d_i, d_j)$$

Document Preprocessing

Document Preprocessing

- Document preprocessing can be divided into five text operations:
  - Lexical analysis of the text
  - Elimination of stopwords
  - Stemming of the remaining words
  - Selection of index terms or keywords
  - Construction of term categorization structures (thesaurus)

Logical View of a Document



Lexical Analysis of the Text

- **Lexical analysis**
  - Process of converting stream of chars into stream of words
  - Major objective: identify words in the text
  - Word separators:
    - **Space:** most common separator
    - **Numbers:** inherently vague, need context for disambiguation
    - **Hyphens:** break up hyphenated words
    - **Punctuation marks:** allows distinguishing **x.id** from **xid** in a program
    - **Case of the letters:** allows distinguishing **Bank** from **bank**

Elimination of Stopwords

- Stopwords
  - words that appear too frequently
  - usually, not good discriminators
  - normally, filtered out as potential index terms
  - natural candidates: articles, prepositions, conjunctions
- Elimination of stopwords
  - reduces size of index by 40% or more
  - at expense of reducing recall: not able to retrieve documents that contain **“to be or not to be”**

Stemming

- User specifies query word but only a variant of it is present in a relevant document
  - plurals, gerund forms, and past tense suffixes
  - partially solved by the adoption of stems
- **Stem**
  - portion of word left after removal of prefixes/suffixes
  - **connect**: stem of *connected, connecting, connection, connections*
- Stemming reduces size of the index
- There is controversy about benefits of stemming for retrieval
- Many search engines do not adopt any stemming

Stemming

- Four types of stemming strategies:
  - **Affix removal**: in which the most important part is suffix removal
    - The Porter algorithm is a suffix removal algorithm for English
  - **Table lookup**: look for the stem of a word in a table
  - **Successor variety**: determine morpheme boundaries and use knowledge from structural linguistics
  - **N-grams**: identify digrams and trigrams (term clustering)

Keyword Selection

- **Full text representation**
  - all words in text used as **index terms** (or, **keywords**)
- Alternative to full text representation
  - Not all words in text used as index terms
  - Case 1: use just nouns as index terms
  - Case 2: group nouns that appear nearby in text into a single indexing component (a concept)
    - **noun groups** as index terms: *computer science, los angeles*
    - logical view of documents as sets of non-elementary index terms

Thesauri

- **Thesaurus**
  - Word has Greek and Latin origins
  - Used as reference to a treasury of words
  - In its simplest form, this treasury consists of
    - precompiled list of important words in a knowledge domain
    - for each word in this list, a set of related words derived from a synonymy relationship

Thesauri

- In general, a thesaurus includes a more complex structure
  - Roget's thesaurus includes **phrases**

    cowardly *adjective*

    Ignobly lacking in courage: *cowardly turncoats*.

    Syns: chicken (slang), chicken-hearted, craven, dastardly, faint-hearted, gutless, lily-livered, pusillanimous, unmanly, yellow (slang), yellow-bellied (slang).
- With the adjective **cowardly**, Roget's thesaurus associates several synonyms which compose a thesaurus class

Thesauri

- The main purposes of a thesaurus are to provide:
  - a standard vocabulary for indexing and searching
  - a means to find terms for proper query formulation
  - classified hierarchies to allow broadening/narrowing queries
- Motivation for building a thesaurus: a **controlled vocabulary** for indexing and searching

Thesauri

- A controlled vocabulary presents important advantages:
  - normalization of indexing concepts
  - reduction of noise
  - identification of indexing terms with a clear semantic meaning
  - retrieval based on concepts rather than on words
- Such advantages are particularly important in specific domains of knowledge

Thesauri

- For general domains
  - a well known body of knowledge that can be associated with the documents in the collection might not exist
  - this might be because collection is new, too large, or changes dynamically
  - thus, not clear how useful a thesaurus is in the context of the Web
- Many search engines simply index **all** the words

Thesaurus Index Terms

- Terms are the **indexing** components of a thesaurus
  - a term can be composed of a word, a group of words, or a phrase
  - it is normally a noun (most concrete part of speech)
  - it usually denotes a **concept**
  - can be expressed as a combination of an adjective with a noun: **polar bear**

Thesaurus Term Relationships

- **Synonyms** and **near-synonyms**
  - Set of terms related to a given thesaurus term
  - Relationships can be induced by patterns of co-occurrence within documents
  - Such relationships are usually of a hierarchical nature
    - **broader** (represented by BT) related terms
    - **narrower** (represented by NT) related terms
  - But, they can also be of a lateral or non-hierarchical nature
    - We simply say that the terms are related (represented by RT)
    - BT and NT relationships can be identified automatically
    - Dealing with RT relationships is much harder

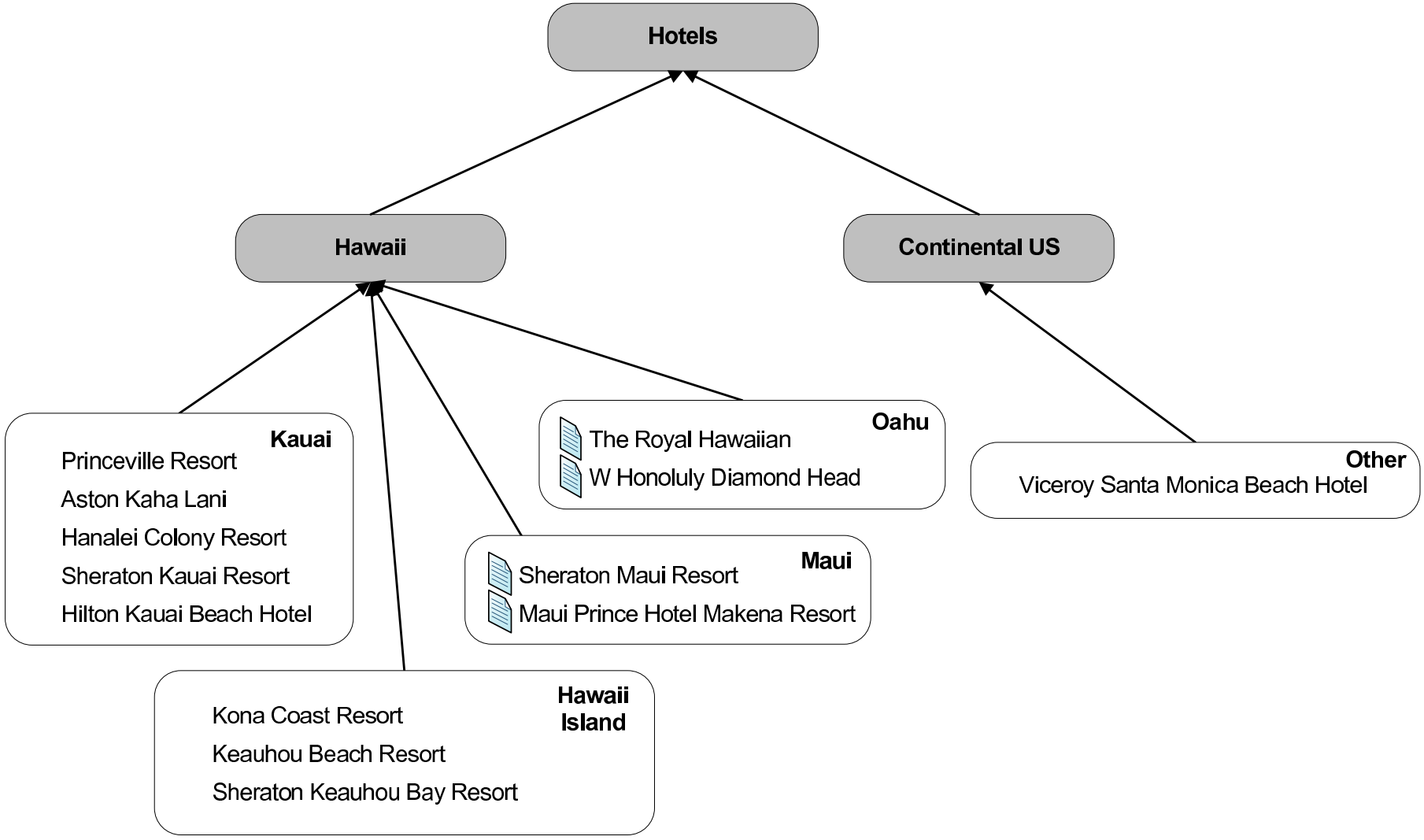
On the Use of Thesauri in IR

- Query formation process
  - User forms a query
  - Query terms might be erroneous and improper
  - Solution: reformulate the original query
  - Usually, this implies expanding original query with related terms
  - Thus, it is natural to use a thesaurus for finding related terms

On the Use of Thesauri in IR

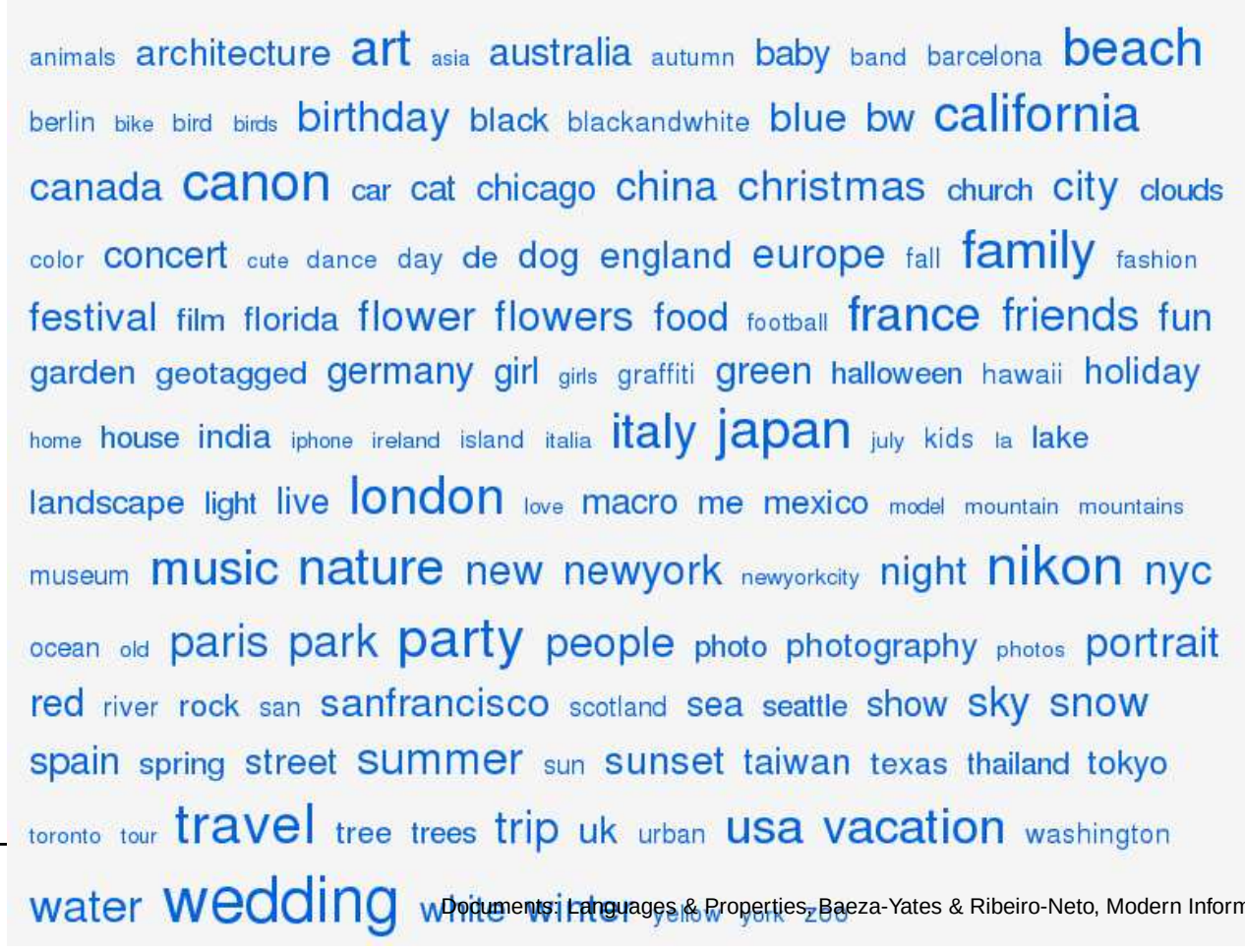
- Relationships captured in a thesaurus are frequently not valid for the local context of a query
- To solve this context problem
  - determine thesaurus-like relationships at query time
  - but, not attractive for search engines
    - expensive in terms of time and resources

Taxonomies


Hotels
Hawaii
Continental US
Kauai
Princeville Resort
Aston Kaha Lani
Hanalei Colony Resort
Sheraton Kauai Resort
Hilton Kauai Beach Hotel
Oahu
The Royal Hawaiian
W Honoluly Diamond Head
Other
Viceroy Santa Monica Beach Hotel
Maui
Sheraton Maui Resort
Maui Prince Hotel Makena Resort
Hawaii Island
Kona Coast Resort
Keauhou Beach Resort
Sheraton Keauhou Bay Resort

Folksonomies

- A **folksonomy** is a collaborative flat vocabulary
  - terms are selected by a population of users
  - each term is called a *tag*



Text Compression

Text Compression

- A representation of text using less space
- Attractive option to reduce costs associated with
  - space requirements
  - input/output (I/O) overhead
  - communication delays
- Becoming an important issue for IR systems
- *Trade-off:* time to encode versus time to decode text

Text Compression

- Our focus are compression methods that
  - allow **random access** to text
  - do not require decoding the entire text
- Important: compression and decompression speed
  - In many situations, decompression speed is more important than compression speed
  - For instance, in textual databases in which texts are compressed once and read many times from disk
- Also important: possibility of searching text without decompressing
  - faster because much less text has to be scanned

Compression Methods

- Two general approaches
  - **statistical** text compression
  - **dictionary** based text compression
- **Statistical methods**
  - Estimate the probability of a symbol to appear next in the text
  - **Symbol**: a character, a text word, a fixed number of chars
  - **Alphabet**: set of all possible symbols in the text
  - **Modeling**: task of estimating probabilities of a symbol
  - **Coding** or **encoding**: process of converting symbols into binary digits using the estimated probabilities

Compression Methods

- **Dictionary methods**
  - Identify a set of sequences that can be referenced
  - Sequences are often called **phrases**
  - Set of phrases is called the **dictionary**
  - Phrases in the text are replaced by pointers to dictionary entries

Statistical Methods

- Defined by the combination of two tasks
  - the **modeling task** estimates a probability for each next symbol
  - the **coding task** encodes the next symbol as a function of the probability assigned to it by the model
- A **code** establishes the representation (**codeword**) for each source symbol
- The entropy E is a **lower bound** on compression, measured in bits per symbol

Statistical Methods

- Golden rule

  Shorter codewords should be assigned to more frequent symbols to achieve higher compression

- If probability p_c of a symbol c is much higher than others, then $\log_2 \frac{1}{p_c}$ will be small

- To achieve good compression

  - Modeler must provide good estimation of probability p of symbol occurrences
  - Encoder must assign codewords of length close to $\log_2 \frac{1}{p}$

Statistical Methods: Modeling

- Compression models can be
  - **adaptive**, **static**, or **semi-static**
  - **character-based** or **word-based**
- *Adaptive models:*
  - start with no information about the text
  - progressively learn the statistical text distribution
  - need only one pass over the text
  - store no additional information apart from the compressed text
- Adaptive models provide an inadequate alternative for full-text retrieval
  - decompression has to start from the beginning of text

Static models

- **Static models**
  - assume an average distribution for all input texts
  - modeling phase is done only once for all texts
  - achieve poor compression ratios when data deviate from initial statistical assumptions
    - a model that is adequate for English literary texts will probably perform poorly for financial texts

Semi-static models

- **Semi-static models**
  - Do not assume any distribution on the data
  - Learn data distribution (fixed code) in a first pass
  - Text compressed in a second pass using fixed code from first pass
  - Information on data distribution sent to decoder before transmitting encoded symbols
  - Advantage in IR contexts: direct access
    - Same model used at every point in compressed file

Semi-static models

- Simple semi-static model: use **global frequency** information
- Let f_c be the frequency of symbol c in the text $T = t_1 t_2 \ldots t_n$
- The corresponding entropy is

$$E = \sum_{c \in \Sigma} \frac{f_c}{n} \log_2 \frac{n}{f_c}$$

- This simple modeling may not capture the redundancies of the text

Semi-static models

- In the 2 gigabyte TREC-3 collection:
  - Entropy under this simple model: 4.5 bits per character
  - Compression ratio cannot be lower than 55%
  - But, state-of-the-art compressors achieve compression ratios between 20% and 40%

Semi-static models

- **Order** k of a model
  - Number of symbols used to estimate probability of next symbol
  - Zero-order model: computed independently of context
  - Compression improves with higher-order models
    - Model of order 3 in TREC-3 collection
      - compression ratios of 30%
      - handling about 1.3 million frequencies
    - Model of order 4 in TREC-3 collection
      - compression ratio of 25%
      - handling about 6 million frequencies
- In adaptive compression, a higher-order modeler requires much more memory to run

Word-based Modeling

- **Word-based modeling** uses zero-order modeling over a sequence of words
- Good reasons to use word-based models in IR
  - Distribution of words more skewed than that of individual chars
  - Number of different words is not as large as text size
  - Words are the atoms on which most IR systems are built
  - Word frequencies are useful in answering queries

Word-based Modeling

- Two different alphabets can be used
  - one for words
  - one for separators
- In TREC-3, 70% – 80% of separators are spaces
- Good properties of word-based models stem from well-known statistical rules:
  - **Heaps' law**: $V = O(n^{\beta})$,
  - **Zipf's law**: the i-th most frequent word occurs $O(n/i^{\alpha})$ times

Statistical Methods: Coding

- **Codeword:** representation of a symbol according to a model
- **Encoders:** generate the codeword of a symbol (coding)
  - assign short codewords to frequent symbols
  - assign long codewords to infrequent ones
  - entropy of probability distribution is lower bound on average length of a codeword
- **Decoders:** obtain the symbol corresponding to a codeword (decoding)
- Speed of encoder and decoder is important

Statistical Methods: Coding

- **Symbol code:** an assignment of a codeword to each symbol
- The least we can expect from a code is that it be **uniquely decodable**
- Consider three source symbols A, B, and C
  - Symbol code: $A \rightarrow$ `0`, $B \rightarrow$ `1`, $C \rightarrow$ `01`
  - Then, compressed text `011` corresponds to ABB or CB?

Statistical Methods: Coding

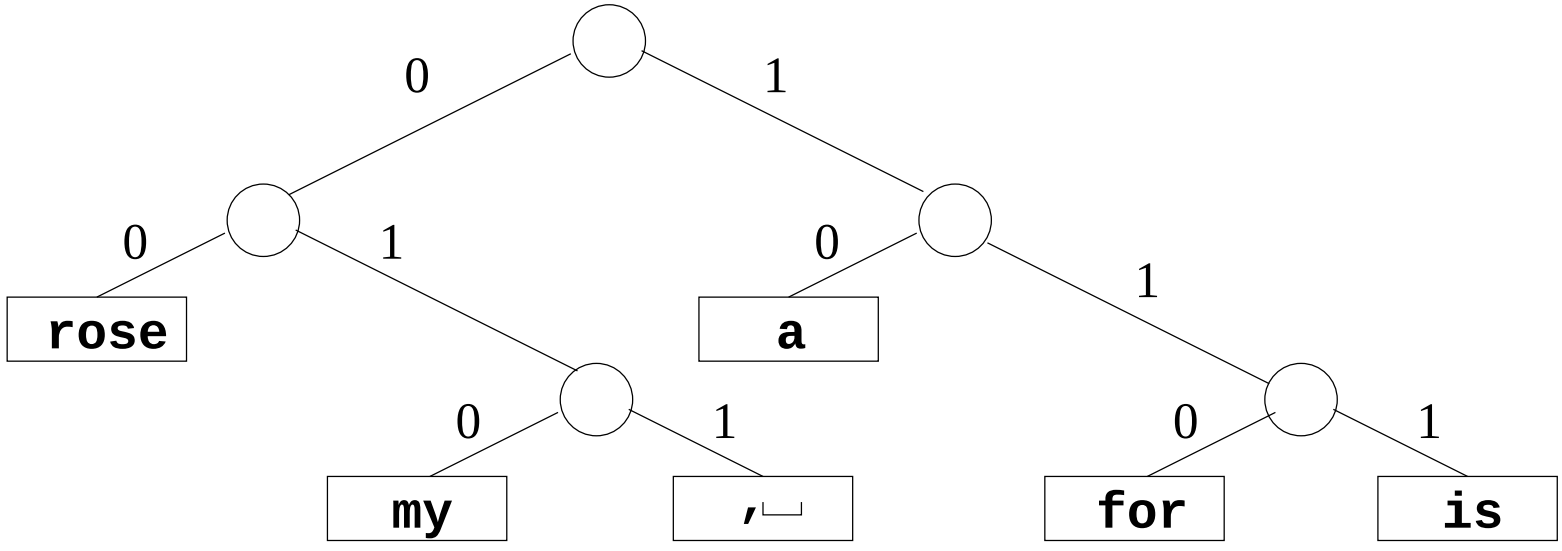
- Consider again the three source symbols A, B, and C
  - Symbol code: $A \rightarrow$ `00`, $B \rightarrow$ `11`, $C \rightarrow$ `110`
  - This symbol code is uniquely decodable
  - However, for the compressed text `110000000`
    - we must count total number of zeros to determine whether first symbol is B or C
- A code is said to be **instantaneous** if every codeword can be decoded immediately after reading its last bit
- **Prefix-free** or **prefix** codes: no codeword should be a prefix of another

Statistical Methods: Coding

- **Huffman coding**
  - a method to find the best prefix code for a probability distribution
  - Let $\{p_c\}$ be a set of probabilities for the symbols $c \in \Sigma$
    - Huffman method assigns to each c a codeword of length ℓ_c
    - Idea: minimize $\sum_{c \in \Sigma} p_c \cdot \ell_c$
  - In a first pass, the modeler of a semi-static Huffman-based compression method:
    - determines the probability distribution of the symbols
    - builds a coding tree according to this distribution
  - In a second pass, each text symbol is encoded according to the coding tree

Huffman Codes

- Figure below presents an example of Huffman compression



Original text: **for my rose, a rose is a rose**

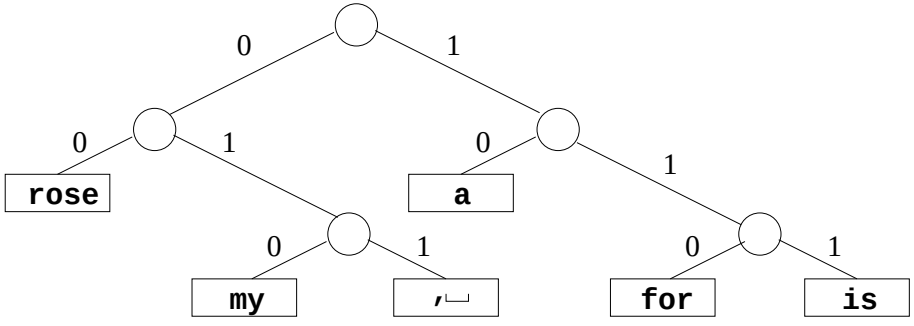
Compressed text: **110 010 00 011 10 00 111 10 00**

Huffman Codes

- Given V symbols and their frequencies in the text, the algorithm builds the Huffman tree in $O(V \log V)$ time
- Decompression is accomplished as follows
  - Stream of bits in file is traversed from left to right
  - Sequence of bits read is used to also traverse the Huffman coding tree, starting at the root
  - Whenever a leaf node is reached, the corresponding word or separator is printed out and the tree traversal is restarted
- In our example, the presence of the codeword `110` in the compressed file leads to the symbol `for`

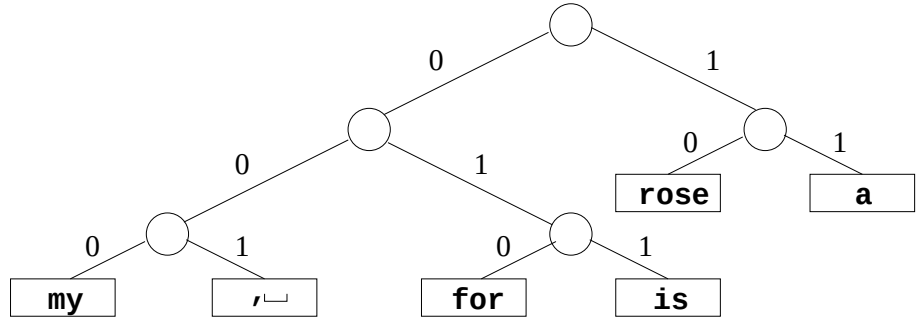
Huffman Codes

- The Huffman tree for a given probability distribution is not unique



Original text: `for my rose, a rose is a rose`

Compressed text: `110 010 00 011 10 00 111 10 00`



Original text: `for my rose, a rose is a rose`

Compressed text: `010 000 10 001 11 10 011 11 10`

- **Canonical tree**
  - right subtree of no node can be taller than its left subtree
  - can be stored very efficiently
  - allows faster decoding

Byte-Huffman Codes

- Original Huffman method leads to binary coding trees
- However, we can make the code assign a sequence of whole bytes to each symbol
  - As a result, Huffman tree has degree 256 instead of 2
  - This word-based model degrades compression ratios to around 30%
  - In exchange, decompression of byte-Huffman code is much faster than for binary Huffman code

Byte-Huffman Codes

- In byte-Huffman coding, direct searching on compressed text is simpler
- To search for a word in the compressed text
  - first find it in the vocabulary
    - for TREC-3, vocabulary requires just 5 megabytes
  - mark the corresponding leaf in the tree
  - proceed over text as if decompressing, except that no symbol is output
  - instead, report occurrences when visiting marked leaves

Byte-Huffman Codes

- Process is simple and fast: only 30% of the I/O is necessary
- Assume we wish to search for a complex pattern including ranges of characters or a regular expression
  - just apply the algorithm over the vocabulary
  - for each match of a whole vocabulary word, mark the word
  - done only on the vocabulary (much smaller than whole text)
  - once relevant words are marked, run simple byte-scanning algorithm over the compressed text

Byte-Huffman Codes

- All complexity of the search is encapsulated in the vocabulary scanning
- For this reason, searching the compressed text is
  - up to 8 times faster when complex patterns are involved
  - about 3 times faster when simple patterns are involved

Byte-Huffman Codes

- Although the search technique is simple and uniform, one could do better especially for single-word queries
- Concatenation of two codewords might contain a third codeword
  - Consider the code: $A \rightarrow$ `0`, $B \rightarrow$ `10`, $C \rightarrow$ `110`, $D \rightarrow$ `111`
    - DB would be coded as `11110`
    - If we search for C, we would incorrectly report a spurious occurrence spanning the codewords of DB
    - To check if the occurrence is spurious or not, rescan all text from the beginning

Dense Codes

- An alternative coding simpler than byte-Huffman is **dense coding**
- Dense codes arrange the symbols in decreasing frequency order
  - Codeword assigned to the i-th most frequent symbol is, essentially, the number $i - 1$
  - number is written in a variable length sequence of bytes
  - 7 bits of each byte are used to encode the number
  - highest bit is reserved to signal the last byte of the codeword

Dense Codes

- Codewords of symbols ranked 1 to 128 are 0 to 127
  - they receive one-byte codewords
  - highest bit is set to 1 to indicate last byte (that is, we add 128 to all codewords)
  - symbol ranked 1 receives codeword $\langle 128 \rangle = \langle 0 + 128 \rangle$
  - symbol ranked 2 receives codeword $\langle 129 \rangle = \langle 1 + 128 \rangle$
  - symbol ranked 128 receives codeword $\langle 255 \rangle$
- Symbols ranked from 129 to 16,512 (i.e., $128 + 128^2$) are assigned two-byte codewords $\langle 0, 128 \rangle$ to $\langle 127, 255 \rangle$

Dense Codes

- **Stoppers**
  - these are those bytes with their highest bit set
  - they indicate the end of the codeword
- **Continuers**
  - these are the bytes other than *stoppers*
- Text vocabularies are rarely large enough to require 4-byte codewords

Dense Codes

- Figure below illustrates an encoding with dense codes

Word rank	Codeword	Bytes	# of words
1	$\langle 128 \rangle$	1	128
2	$\langle 129 \rangle$	1	
...	...		
128	$\langle 255 \rangle$	1	
129	$\langle 0, 128 \rangle$	2	128^2
130	$\langle 0, 129 \rangle$	2	
...	...		
256	$\langle 0, 255 \rangle$	2	
257	$\langle 1, 128 \rangle$	2	
...	...		
16,512	$\langle 127, 255 \rangle$	2	
16,513	$\langle 0, 0, 128 \rangle$	3	128^3
...	...		
2,113,664	$\langle 127, 127, 255 \rangle$	3	

Dense Codes

- Highest bit signals the end of a codeword
  - a dense code is automatically a prefix code
- **Self-synchronization**
  - Dense codes are **self-synchronizing**
    - Given any position in the compressed text, it is very easy to determine the next or previous codeword beginning
    - decompression can start from any position, be it a codeword beginning or not
  - Huffman-encoding is **not self-synchronizing**
    - not possible to decode starting from an **arbitrary position** in the compressed text
    - notice that it is possible to decode starting at an *arbitrary codeword beginning*

Dense Codes

- Self-synchronization allows faster search
- To search for a single word we can
  - obtain its codeword
  - search for the codeword in the compressed text using **any** string matching algorithm
- This does not work over byte-Huffman coding

Dense Codes

- An *spurious occurrence* is a codeword that is a suffix of another codeword
  - assume we look for codeword a b $\overline{\text{c}}$, where we have overlined the stopper byte
  - there could be a codeword d a b $\overline{\text{c}}$ in the code, so that we could find our codeword in the text . . . e f $\overline{\text{g}}$ d a b $\overline{\text{c}}$ . . .
  - yet, it is sufficient to access the text position preceding the candidate occurrence, ‘d’, to see that it is not a stopper
- Such a fast and simple check is not possible with Huffman coding
- To search for phrases
  - concatenate the codewords
  - search for the concatenation

(s, c)-Dense Codes

- Dense codes assign
  - values 0 to 127 to continuer bytes
  - values 128 to 255 to stopper bytes
  - split is arbitrary and could be changed to c continuers and s stoppers, for $s + c = 256$
- Stoppers and continuers can be distinguished by considering the numerical value of the bytes
  - s most frequent symbols are encoded with 1 byte
  - next sc most frequent symbols are encoded with 2 bytes
  - next sc^2 most frequent symbols are encoded with 3 bytes
- For the TREC collection,
  - optimal values for s: between 185 and 200
  - compression ratios very close to those of byte-Huffman codes

(s, c)-Dense Codes

- Finding the optimal s and c values is not expensive
  - sort the V symbols by decreasing frequencies f_i
  - compute the cumulative frequencies in a single pass
    - $F_0 = 0$
    - $F_i = F_{i-1} + f_i$
  - total compressed file length using a given (s, c) combination is

$$1 \times F_s \; + \; 2 \times (F_{s+sc} - F_s) \; + \; 3 \times (F_{s+sc+sc^2} - F_{s+sc}) \; + \; \ldots$$

  - sum has $O(\log_c V)$ terms and can be easily recomputed for all possible 256 (s, c) combinations

(s, c)-Dense Codes

- Dense codes compared to Huffman codes
  - are much simpler to handle
    - no coding tree has to be built nor stored — just the sorted vocabulary suffices
    - simple algebraic calculation to convert word ranks into codes, and vice versa
  - yield almost the same compression ratios (as byte-Huffman)
  - permit much faster searching
  - are simpler and faster to program and manipulate
  - have stronger synchronization properties
  - share all the positive features of Huffman coding

Dictionary Methods

- **Dictionary methods**
  - identify groups of consecutive symbols (or phrases) that can be referenced
  - occurrences of the phrases in the text are replaced by a pointer to an entry in a dictionary
  - choice of phrases can be made by **static**, **semi-adaptive**, or **adaptive** algorithms

Dictionary Methods

- **Static dictionary** encoders are fast
  - little effort for a small amount of compression
  - Example: digram coding in which selected pairs of letters are replaced with special characters
- They are rarely very effective
  - a dictionary might be suitable for one text and unsuitable for another

Dictionary Methods

- **Adaptive dictionary** methods are more successful
  - in Ziv-Lempel compression, the dictionary is built as the text is compressed
  - depending on the Ziv-Lempel compressor, different substrings can be referenced
- Ziv-Lempel methods are very popular as general-purpose compressors
  - implemented in various compression softwares: `zip`, `pkzip`, `gzip`, `winzip`, `arj`
  - they do not allow access to the compressed text at random positions

Re-Pair Methods

- Much more interesting for IR is **Re-Pair**:
  - assign an integer to each symbol in the alphabet
  - rewrite the text as the corresponding sequence of integers
  - iteratively, identify the most frequent pair of consecutive integers,
  - insert into the dictionary a rule $C \rightarrow A \cdot B$, for a fresh integer C
  - replace all the occurrences of $A \cdot B$ to C
- Repeat this process until no consecutive pair repeats in the text

Re-Pair Methods

- A phrase like C can participate in new phrases like $E \rightarrow D \cdot C$
  - dictionary can be seen as a binary hierarchy of phrases
- Advantages:
  - Re-Pair obtains very good compression ratios
  - It can easily decompress the text starting at any position
- Drawbacks:
  - compression is extremely slow
  - compression needs large amounts of memory

Re-Pair Methods

- There is, however, a version of Re-Pair of utmost interest for IR applications:
  - take the text as a sequence of words
  - apply the same algorithm over the resulting sequence of integers
  - compression ratios worsen a bit
  - but, all phrases in the dictionary are sequences of words
  - **frequent phrases** can be used for **phrase browsing**
    - user writes `'United'` and sees related phrases in the text: `'United Nations'`, `'United States'`, and `'Manchester United'`
- Re-Pair allows searching compressed text for words
  - find all dictionary phrases than contain the word
  - search for all those phrases simultaneously

Preprocessing for Compression

- **Preprocessing for compression**:
  - transform the text to a form that is easier to compress
  - for instance, **Burrows-Wheeler Transform (BWT)** performs a permutation of the positions of a text
- The BWT of a text $T = t_1 t_2 \ldots t_n$ is defined as follows
  - Assume $t_n = \$$ is a special terminator character
  - Form a conceptual matrix M whose rows are all the **cyclic shifts** of T, $t_i t_{i+1} \ldots t_n t_1 t_2 \ldots t_{i-1}$ for all i
  - Sort the rows of M lexicographically
  - Call F the first column of M and L the last column
  - The BWT of T is L

Preprocessing for Compression

- Building the Burrows-Wheeler Transform of ‘mississippi$’

m	i	s	s	i	s	s	i	p	p	i	$
i	s	s	i	s	s	i	p	p	i	$	m
s	s	i	s	s	i	p	p	i	$	m	i
s	i	s	s	i	p	p	i	$	m	i	s
i	s	s	i	p	p	i	$	m	i	s	s
s	s	i	p	p	i	$	m	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
i	p	p	i	$	m	i	s	s	i	s	s
p	p	i	$	m	i	s	s	i	s	s	i
p	i	$	m	i	s	s	i	s	s	i	p
i	$	m	i	s	s	i	s	s	i	p	p
$	m	i	s	s	i	s	s	i	p	p	i

Preprocessing for Compression

- Building the Burrows-Wheeler Transform of ‘mississippi$’

F											L
$	m	i	s	s	i	s	s	i	p	p	i
i	$	m	i	s	s	i	s	s	i	p	p
i	p	p	i	$	m	i	s	s	i	s	s
i	s	s	i	p	p	i	$	m	i	s	s
i	s	s	i	s	s	i	p	p	i	$	m
m	i	s	s	i	s	s	i	p	p	i	$
p	i	$	m	i	s	s	i	s	s	i	p
p	p	i	$	m	i	s	s	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
s	i	s	s	i	p	p	i	$	m	i	s
s	s	i	p	p	i	$	m	i	s	s	i
s	s	i	s	s	i	p	p	i	$	m	i

Preprocessing for Compression

- Somewhat surprisingly, it is possible to recover T from L
- Why BWT should help compression
  - Consider a common phrase: `'United States'`
  - Occurrences of `'ed States'` in T appear contiguously in M
  - In most cases, `'ed States'` is preceded by `'t'`
  - L will contain mostly `'t'`s in contiguous area
  - Similarly, `'ted States'`, `'ited States'`, and `'nited States'` will produce long **runs** of letters `'i'`, `'n'`, and `'U'`, respectively, in L
- Key point of BWT: k-th order redundancies in T translate into runs of few different characters in L
- Widespread compressor based on BWT: **bzip2**

Comparing Techniques

- Statistical methods:
  - Semi-static methods: **bit-Huffman** and **byte-dense**
  - Adaptive methods: **PPM** (program `ppmdi`)
- Dictionary methods:
  - Semi-static method: **Re-Pair**
  - Adaptive method: **LZ77** of the Ziv-Lempel family
- Method based on text preprocessing: **BWT** (`bzip2` program)

Comparing Techniques

- Comparison of the main compression techniques

Measure	Huffman	Dense	PPM	Re-Pair	LZ77	BWT
Compression ratio (%)	25–30	30–35	20–25	20–30	30–40	25–30
Compression speed	fast	fast	slow	very slow	fast	slow
Decompression speed	fast	very fast	slow	fast	very fast	slow
Memory space	high	high	high	high	low	high
Direct search	fast	very fast	no	fast	slow	no
Random access	yes	yes	no	yes	no	no

Structured Text Compression

- Structured text is a common form of representing documents in many IR systems
- It is possible to use the structure to achieve improved compression
- We opt for mentioning the most prominent products and prototypes for structured text compression

XMLPPM

- Relies on PPM-like modeling and arithmetic coding
- Is adaptive and does not permit direct access to the compressed file
- Several models are active at the same time, and XMLPPM switches between them
- Four models are used:
  - one for element and attribute names
  - one for element structure
  - one for attributes
  - one for strings

XMLPPM

- For example, consider the source text

```
<author><title>Prof.</title>Claude
Shannon</author>
```

- Then,
  - The order-2 PPM context to model ‘Prof.’ will be ‘<author> <title>’
  - The context to model ‘Claude’ will be ‘<author>’
  - The context to model ‘Shannon’ will be ‘<author> Claude’

SCM: Structured Context Modeling

- Simplification of XMLPPM
  - direct access is possible
  - model and encode text under each tag separately from the rest
- For example, in an email archive
  - fields like `<from>` and `<to>` contain email addresses
  - `<date>` contains dates
  - `<subject>` and `<body>` contain free text
- SCM
  - uses a word-based bit-Huffman encoder for each tag
  - merges tags when the distributions turn out to be similar
  - over TREC-3, achieves 2% – 4% improvement in compression compared to word-based bit-Huffman

LZCS

- LZCS (Lempel-Ziv to Compress Structure)
  - aims at very repetitive structured collections
  - archives of Web forms, e-commerce documents, Web service exchange documents
- The method replaces any subtree of the structure by a pointer to an identical subtree seen before
- The references point to positions in the compressed file, not the uncompressed file
- As a consequence, it is easy to navigate the document in compressed form without decompressing it at all