Søketeknologi kompendium

Ole Kristian Stumpf

June 2, 2014

Contents

1 Lecture 1 Boolean retrieval 7
  1.1 Information Retrieval .......................... 7
  1.2 Index ........................................... 7
  1.3 Terms ........................................... 8
  1.4 Boolean retrieval model ...................... 8
  1.5 Collection (Corpus) .......................... 8
  1.6 Ad hoc retrieval ................................ 8
  1.7 Effectiveness ................................. 8
    1.7.1 Precision ................................. 8
    1.7.2 Recall .................................... 8
  1.8 Inverted index (II, inverted file) .......... 8
  1.9 Posting ....................................... 8
  1.10 Posting lists ................................ 9
  1.11 Postings ..................................... 9
  1.12 Dictionary ................................... 9
  1.13 Sorting ....................................... 9
  1.14 Posting lists intersection (merging posting list) ........................................... 9
  1.15 Query optimization ......................... 9
  1.16 Ranked retrieval models ................... 10
  1.17 Proximity operators ......................... 10
  1.18 Term frequency ............................. 10

2 Lecture 2 The term vocabulary and postings lists (PLs) 10
  2.1 Indexing granularity ......................... 10
  2.2 Token ......................................... 10
  2.3 Type .......................................... 10
  2.4 Stop words ................................... 10
2.5 Collection frequency .............................................. 10
2.6 Case folding ......................................................... 11
2.7 Truecasing .......................................................... 11
2.8 Stemming ............................................................ 11
2.9 Lemmatization ...................................................... 11
2.10 Lemmatizer .......................................................... 11
2.11 Skip lists ............................................................. 11
2.12 Biword index ......................................................... 11
2.13 Phrase index ........................................................ 12
2.14 Positional index ..................................................... 12

3 Lecture 3 Dictionaries and tolerant retrieval + selected papers 12
3.1 Wildcard query ...................................................... 12
3.2 Binary tree .......................................................... 12
3.3 B-tree ................................................................. 12
3.4 Search tree vs HashMap as data structure for the dictionary ... 12
  3.4.1 Hashing ........................................................... 13
  3.4.2 Trees .............................................................. 13
3.5 Trailing wildcard query ............................................. 13
  3.5.1 Ex ................................................................. 13
3.6 Leading wildcard query ............................................ 13
  3.6.1 Ex ................................................................. 13
3.7 Infix wildcard query ............................................... 14
  3.7.1 Ex ................................................................. 14
3.8 Permuterm index ................................................... 14
  3.8.1 Ex ................................................................. 14
  3.8.2 Ex ................................................................. 14
3.9 Permuterm vocabulary .............................................. 14
3.10 k-Gram index ....................................................... 15
  3.10.1 Ex ................................................................. 15
  3.10.2 Ex ................................................................. 15
3.11 Edit distance/Levenshtein distance .............................. 15
3.12 Implementing spelling correction ................................ 15
3.13 Jaccard coefficient ............................................... 16
3.14 Tries hentet fra wikipedia (← link) ............................. 16
  3.14.1 Pros .............................................................. 16
  3.14.2 Cons .............................................................. 17
3.15 Aho cohasic fra wikipedia ...................................... 17
# Lecture 4 Index construction (Indexing)

## 4.1 Indexer

## 4.2 Index construction techniques

### 4.2.1 Blocked sort-based indexing (BSBI)

### 4.2.2 Single-pass in-memory indexing (SPIMI)

### 4.2.3 Distributed indexing (MapReduce)

### 4.2.4 Dynamic indexing

### 4.2.5 Other types of indexes

# Lecture 5 Index compression + selected papers

## 5.1 Lossless compression

## 5.2 Lossy compression

## 5.3 Heap’s law

### 5.3.1 Suggests:

## 5.4 Zips’s law

## 5.5 Front coding

## 5.6 Variable byte codes

### 5.6.1 Examples

## 5.7 Gamma codes

### 5.7.1 Examples

## 5.8 Simple9 (S9) coding

### 5.8.1 Example

## 5.9 Universal code

## 5.10 Prefix free

## 5.11 Parameter free

## 5.12 Rice coding

### 5.12.1 Example

### 5.12.2 Example

### 5.12.3 Example

# Lecture 6 Scoring, term weighting, and vector space model

## 6.1 Metadata

## 6.2 Field

## 6.3 Parametric indexes

## 6.4 Zones

## 6.5 Weighted zone scoring

## 6.6 Term frequency

## 6.7 Bag of words

## 6.8 Document frequency

## 6.9 Inverse document frequency

## 6.10 Tf-idf
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.3</td>
<td>XML element</td>
<td>30</td>
</tr>
<tr>
<td>10.4</td>
<td>XML attribute</td>
<td>30</td>
</tr>
<tr>
<td>10.5</td>
<td>XML DOM</td>
<td>30</td>
</tr>
<tr>
<td>10.6</td>
<td>XML Context/contexts</td>
<td>30</td>
</tr>
<tr>
<td>10.7</td>
<td>Schema</td>
<td>30</td>
</tr>
<tr>
<td>10.8</td>
<td>NEXI</td>
<td>31</td>
</tr>
<tr>
<td>10.9</td>
<td>Structured document retrieval principle</td>
<td>31</td>
</tr>
<tr>
<td>10.10</td>
<td>Indexing unit</td>
<td>31</td>
</tr>
<tr>
<td>10.11</td>
<td>Schema heterogeneity/diversity</td>
<td>31</td>
</tr>
<tr>
<td>10.12</td>
<td>Extended query</td>
<td>31</td>
</tr>
<tr>
<td>10.13</td>
<td>Structural term</td>
<td>31</td>
</tr>
<tr>
<td>10.14</td>
<td>Text-centric XML</td>
<td>31</td>
</tr>
<tr>
<td>10.15</td>
<td>Data-centric XML</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Lecture 11 Probabilistic information retrieval</td>
<td>32</td>
</tr>
<tr>
<td>11.1</td>
<td>Probability ranking principle (RPR)</td>
<td>32</td>
</tr>
<tr>
<td>11.2</td>
<td>1/0 loss</td>
<td>32</td>
</tr>
<tr>
<td>11.3</td>
<td>Bayes optimal rule</td>
<td>32</td>
</tr>
<tr>
<td>11.4</td>
<td>Binary independence model</td>
<td>33</td>
</tr>
<tr>
<td>11.5</td>
<td>Naive Bayes assumption</td>
<td>33</td>
</tr>
<tr>
<td>11.6</td>
<td>Retrieval status value</td>
<td>33</td>
</tr>
<tr>
<td>11.7</td>
<td>Relative frequency</td>
<td>33</td>
</tr>
<tr>
<td>11.8</td>
<td>BMI25 weights/Okapi weighting</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Lecture 12 [TOMT]</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Lecture 13 Text classification and Naive Bayes</td>
<td>33</td>
</tr>
<tr>
<td>13.1</td>
<td>Standing query</td>
<td>33</td>
</tr>
<tr>
<td>13.2</td>
<td>Classification</td>
<td>33</td>
</tr>
<tr>
<td>13.3</td>
<td>Routing/filtering</td>
<td>34</td>
</tr>
<tr>
<td>13.4</td>
<td>Topics</td>
<td>34</td>
</tr>
<tr>
<td>13.5</td>
<td>Applications of classification in IR</td>
<td>34</td>
</tr>
<tr>
<td>13.5.1</td>
<td>Preprocessing steps</td>
<td>34</td>
</tr>
<tr>
<td>13.5.2</td>
<td>Spam</td>
<td>34</td>
</tr>
<tr>
<td>13.5.3</td>
<td>Porn mode</td>
<td>34</td>
</tr>
<tr>
<td>13.5.4</td>
<td>Sentiment detection</td>
<td>34</td>
</tr>
<tr>
<td>13.5.5</td>
<td>Personal email sorting</td>
<td>34</td>
</tr>
<tr>
<td>13.5.6</td>
<td>Topic-specific or vertical search</td>
<td>34</td>
</tr>
<tr>
<td>13.5.7</td>
<td>Ranking function</td>
<td>35</td>
</tr>
<tr>
<td>13.6</td>
<td>Rules in text classification (TC)</td>
<td>35</td>
</tr>
</tbody>
</table>
Lecture 1 Boolean retrieval

Merk: Har ikke skrevet om How to squeeze a lexicon + kap 9.

Information Retrieval

IR is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)

Index

The way to avoid linearly scanning the text for each query is to index the docs in advance
Terms
Are the indexed units

Boolean retrieval model
Is a model for information retrieval in which we can pose any query which is in the form of a Boolean expression of terms, that is, in which terms are combined with the operators AND, OR and NOT. The model views each document as just a set of words.

Collection (Corpus)
Group of documents over which we perform retrieval task.

Ad hoc retrieval
An IR task. In it, a system aims to provide documents from within the collection that are relevant to an arbitrary user information need.

Effectiveness
Two key factors

Precision
What fraction of the returned results are relevant to the information need?

Recall
What fraction of the relevant documents in the collection were returned by the system?

Inverted index (II, inverted file)
We keep a dictionary of terms. For each term, we have a list that records which documents the term occurs in.

Posting
Each item in a II records the docID of the term.
Posting lists
The list of postings is a posting list

Postings
All posting lists put together.

Dictionary
All terms in the II put together

Sorting
Core idea of II. Terms must be alphabetically ordered.

Posting lists intersection (merging posting list)
Must intersect postings lists to find docs that contain both terms Common
intersection algorithm:

1. Maintain pointers to each PL, starting at the beginning
2. Compare the docID pointed to by the pointers
3. If they are the same, put posting in result list and advance both point-
ers
4. Otherwise, advance the pointer pointing to the smaller docID and re-
peat from step 2.

O(x + y) operations where x and y are the length of the postinglist, but
formally the complexity is O(N) where N is the size of the corpus.
Important! To use this algorithm, the PLs must be sorted by a single
global ordering.

Query optimization
Is the process of selecting how to organize the work of answering a query so
that the least total amount of work needs to be done by the system
Ranked retrieval models
The boolean retrieval model contrasts with ranked retrieval models such as
the vector space model, in which the system decides which documents best
satisfy the query.

Proximity operators
Is a way of specifying that two terms in a query must occur close to each
other in the document. More info at Lecture 2: Positional index

Term frequency
The number of times a term occurs in a document

Lecture 2 The term vocabulary and postings lists (PLs)

Indexing granularity
For a given collection, what is the size of each document? For a collection of
books, it would usually be a bad idea to index an entire book as a document.

Token
Is an instance of a sequence of characters in some particular document that
are grouped together as useful semantic for processing.

Type
Is the class of all tokens containing the same character sequence.

Stop words
Extremely common words that would appear to be of little value in helping
select the docs matching a used need.

Collection frequency
Total number of times each term appears in the document collection
**Case folding**
Reducing all letters to lower case

**Truecasing**
Machine learning on when to do case folding

**Stemming**
Refers to a crude heuristic process that chops off the ends of words. Ex: Saw → s

**Lemmatization**
Refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. Ex: Saw → see/saw (depending on whether the word is used as a noun or as a verb)

**Lemmatizer**
A tool that finds the lemma in words.

**Skip lists**
A technique to merge posting lists in sublinear time. Augment the PLs with skip pointers that function as shortcuts that allow us to avoid processing parts of the posting list that will not figure in the search result.

**Biword index**
Technique for handling phrase queries. Consider every pair of consecutive terms in a document as a phrase. Ex: Friends, Romans, Countrymen become the biwords “friends romans” and “romans countrymen”. Each biword is treated as a vocabulary term and we index every biword. Longer phrase queries is broken down with AND as connector. Can cause false positives. Consider the example: Query: Standford University Palo Alto → stanford university AND university palo AND palo alto. If this returns a match, there is still no guarantee that the document contains the phrase stanford university Palo Alto.
Phrase index

Extension of biword index. (More than two words)

Positional index

Technique for handling phrase queries. A posting will be on the form: docID:<pos1, pos2..> To process a phrase query, you still need to index the II entries for each term. You start with the least frequent term and then work to further restrict the list of possible candidates. In the merge algorithm, you cannot just check if each term is in the same doc, but you also need to check that their positions of appearance in the document are compatible with the phrase query being evaluated.

The same general method is applied for within k word proximity searches (proximity operators).

Lecture 3 Dictionaries and tolerant retrieval + selected papers

Wildcard query

Ex: a*e*i*o*u, which seeks documents containing any term that includes all the five vowels in sequence. * indicates any (possibly empty) string of characters.

Binary tree

A data structures for storing a dictionary. (Must be balanced).

B-tree

A search tree in which every internal node usually has between 2 and 4 children. In binary tree and b-tree, there has to be an internal ordering of words. Some Asian language do not always have a unique ordering.

Search tree vs HashMap as data structure for the dictionary

Governed by 3 or 4 questions:

1. How many keys (entrie in the vocabulary) are we likely to have
2. Is this number likely to remain static or dynamic?
   - If dynamic, will keys often only be deleted or added?

3. What are the relative frequencies with which various keys will be accessed.

Hashing
- Each term is a key and hashed into an integer over a large enough space that hash collisions are unlikely.
- Cons:
  - Collisions are resolved by auxiliary structures that demand care to maintain.
  - At query time, we hash each query term separately, and, following a pointer to the corresponding postings, taking into account any logic for resolving hash collision.
  - We cannot seek to find minor variants of terms, as these would be hashed to a very different integer.

Trees
Overcome many of these issues. For instance, they permit us to enumerate all vocabulary terms beginning with automat. Best known tree: Binary tree

**Trailing wildcard query**
A query where the * is at the end of the query word.

**Ex:**
mon*. These queries are easy to handle with a tree structure. We walk down the tree following the symbols m, o, n in turn at which we can enumerate the set of terms in the dictionary with the prefix mon.

**Leading wildcard query**
**Ex:**
*mon. Consider a reverse B-tree, one in which each root-to-leaf path of the tree corresponds to a term in the dictionary written backwards. Thus the
term lemon would be represented by the path root n o m e l. As before, you walk down the reverse B-tree then enumerate all terms in the vocabulary.

**Infix wildcard query**

**Ex**

se*mon. First, search for se* with a non-empty suffix. Then, the reverse B-tree enumerates the set of terms ending with the suffix mon. Next, we take the intersection of the results to arrive at the set of terms that begin with the prefix se and end with the suffix mon. We scan and filter out any terms that match the prefix as well as the suffix because these two strings overlap.

**Permuterm index**

Technique for handling general wildcard queries. A form of II. $ indicates the end of a term. To construct a permuterm index, we link all the rotations of a term to the original term. Therefore hello$, ello$h, llo$he, lo$hel all link to hello.

**Ex**

Query: m*n. Rotate it so that the * appears at the end of the string. Therefore: n$m*. Next, we look up this string in the permuterm index, where seeking n$m* (via a search tree) leads to rotations of (among others) the terms man and moron. Now that the permuterm index enables us to indentify the original vocabulary terms matching a wildcary query, we look up these terms in the standard II to retrieve matching docs.

**Ex**

Query: fi*mo*er. First enumerate the terms in the dictionary that are in the permuterm index of er$fi*. Not all terms have mo in the middle so we do an exhaustive search to filter these out. We then run the surviving terms through the standard II for doc retrieval.

**Permuterm vocabulary**

Set of rotated terms in a permuterm index
**k-Gram index**

Technique for handling general wildcard queries. In a k-gram index, the dictionary contains all k-grams that occur in any term in the vocabulary. Each PL points from a k-gram to all vocabulary terms containing that k-gram. For instance, the 3-gram etr would point to terms such as metric and retrieval.

**Ex**

Query re*ve. Run the Boolean query $re \text{AND} ve$. This is looked up in the 3-gram index and yields a list of matching terms such as relive, remove and retrieve. Each of these matching terms is then looked up in the standard II index to yield documents matching the query.

**Ex**

Difficulty: Consider using the 3-gram index described for the query red*. We first issue the boolean query $re \text{AND} RED$ to the 3-gram index. This leads to match terms such as retired, which is not a match. To cope with this, we introduce a postfiltering step, in which the terms are checked individually against the original query.

**Edit distance/Levenshtein distance**

The edit distance between them is the minimum number of edit operations required to transform a string to a different string.

**Implementing spelling correction**

There are two basic principles underlying most spelling correction algorithms:

1. Of various alternative correct spellings for a misspelled query, choose the “nearest” one. This demands that we have a notion of nearness or proximity between a pair of queries. (Edit distance)

2. When to correctly spelled queries are tied, select the one that is more common. For instance, if grunt and grant both both seem equally plausible as correction for grnt. Then the algorithm should choose the more common of grunt and grant as the correction.
Jaccard coefficient

A formula for measuring the overlap between two sets A and B.

\[
\frac{A \cap B}{A \cup B}
\]

The two sets we consider are the set of k-grams in the query q and the set of k-grams in a vocabulary term. As we proceed from one vocabulary term to the next next, computing the Jaccard coefficient on the fly. If the coefficient exceeds a preset threshold, we add the term to the output.

Tries hentet fra [wikipedia](<– link)]

Is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string.

Pros:

1. Looking up data in a trie is faster in the worst case, \(O(m)\) time (where \(m\) is the length of a search string), compared to an imperfect hash table. An imperfect hash table can have key collisions. A key collision is the hash function mapping of different keys to the same position in a hash table. The worst-case lookup speed in an imperfect hash table is \(O(N)\) time, but far more typically is \(O(1)\), with \(O(m)\) time spent evaluating the hash.

2. There are no collisions of different keys in a trie.

3. Buckets in a trie, which are analogous to hash table buckets that store key collisions, are necessary only if a single key is associated with more than one value.

4. There is no need to provide a hash function or to change hash functions as more keys are added to a trie.

5. A trie can provide an alphabetical ordering of the entries by key.
Cons

1. Tries can be slower in some cases than hash tables for looking up data, especially if the data is directly accessed on a hard disk drive or some other secondary storage device where the random-access time is high compared to main memory.

2. Some keys, such as floating point numbers, can lead to long chains and prefixes that are not particularly meaningful. Nevertheless, a bitwise trie can handle standard IEEE single and double format floating point numbers.

3. Some tries can require more space than a hash table, as memory may be allocated for each character in the search string, rather than a single chunk of memory for the whole entry, as in most hash tables.

Aho cohasic fra wikipedia

Is a kind of dictionary-matching algorithm that locates elements of a finite set of strings (the “dictionary”) within an input text. It matches all patterns simultaneously. Informally, the algorithm constructs a finite state machine that resembles a trie with additional links between the various internal nodes. These extra internal links allow fast transitions between failed pattern matches (e.g. a search for cat in a trie that does not contain cat, but contains cart, and thus would fail at the node prefixed by ca), to other branches of the trie that share a common prefix (e.g., in the previous case, a branch for attribute might be the best lateral transition). This allows the automaton to transition between pattern matches without the need for backtracking. Look at the example at the wikipedia page.

Lecture 4 Index construction (Indexing)

Indexer

The machine that performs the index construction

Index construction techniques

With not enough main memory (MM), we need to use an external sorting algorithm, that is, one that uses disk. Must minimize the number of random disk seeks during sorting.
Blocked sort-based indexing (BSBI)

- Steps
  1. Accumulate terms and docIDs into MM and “convert” terms into termID until a fixed block size is reached.
  2. Invert it (convert the block into an inverted index)
  3. Store intermediate sorted results in disk
  4. Merge all intermediate results into the final index

Single-pass in-memory indexing (SPIMI)

BSBI has excellent scaling properties, but needs a data structure for mapping terms to termIDs. For a very large collection, this data structure does not fit into memory. Therefore, use SPIMI

- Steps
  Best to look at the pseudo-code on page 67. Essentially, it adds terms to PLs. Because we do not know how large the PL will be when we create it, we allocate space for a short PL initially, and then double the space each time it is full. When there is no more free memory available, we write the index of the block to disk, after we sort the terms. Then, as in BSBI we merge blocks into the final index.

Distributed indexing (MapReduce)

When collections are too large to fit on a single machine (Web search). A master node directs the process of assigning and reassigning tasks to individuals worker nodes (machines).

- Steps
  1. The input data are split into n splits where the size of the split is chosen to ensure that the work can be distributed evenly and efficiently.
  2. Map phase: Consists of mapping splits of the input data to key-value pairs. (termID, docID). Each parser writes its output to a local intermediate file, the segment files.
  3. Reduce phase: Collecting all values (here: docID) for a given key (termID), into one list is the task of the inverters in the reduce phase.
4. The list of values is sorted for each key and written to the final sorted postings lists.

Remember, parsers and inverters are not separate sets of machines. The master identifies idle machines and assigns tasks to them. This was the design of the Google indexing system in 2004, however it was much more complex.

Dynamic indexing

What if the document collection is dynamic? Auxiliary index: Store two indexes, a large main index and a small auxiliary index that stores new docs. The aux. index is kept in MM. Searches are run across both indexes and results are merged. Deletions are stored in a bit vector, we can then filter out deleted docs before returning the search results. When the aux. index is too large, we merge it with the main index.

Other types of indexes

Detta gider jeg ikke

Lecture 5 Index compression + selected papers

Lossless compression

All information is preserved

Lossy compression

All information is not preserved, but better compression rates can be accomplished. Case folding, stemming and stop word elimination are form of lossy compression.

Heap’s law

\[ M = kT^b \]

\( M \) = number of distinct terms in a collection \( T \) = number of tokens in the collection \( k \) = constant. Usually \( 30 \leq k \leq 100 \). Varies because vocabulary growth depends a lot on the nature of the collection and how it is processed. (Case-folding, stemming reduces the growth rate, but including numbers and spelling errors increase it). \( b \) = constant. Usually roughly 0.5. Is linear in log-log space.
Suggests:

1. The dictionary size continues to increase with more documents in the collection rather than maximum vocabulary size being reached.

2. The size of the dictionary is quite large for large collections.

Zips’s law

Helps us to understand how terms are distributed across docs. This helps us to characterize the properties of the algorithms for compressing postings lists in a later section. \( CF_i = c i^k \). The collection frequency \( CF_i \) of the \( i \)th most common term is \( c \) (some constant) times \( i \) raised to the power of \( k \) with \( k = -1 \). So if the most frequent terms occurs \( CF_1 \) times, then the second most frequent terms has half as many occurrences, the third most frequent terms a third as many occursces and so on...

Front coding

Compression technique where the dictionary is stored as a long string and where a common prefix is identified for a subsequence of the term list and then referred to with a special character.

Variable byte codes

Technique for compressing PLs. First, store the gap from each docIDs to the next. First bit for every byte: continuation bit.

Examples

Gap:

\[
\begin{array}{cccc}
5 & 214577 \\
10000101 & 00001101 & 00001100 & 10110001 \\
\end{array}
\]

We see that as long as there are bytes to be read (from left to right) the first bit of every byte is 0. If its the last byte, its 1.

Gamma codes

Technique for compressing PLs. Implements variable length encoding by splitting the representation of a gap \( G \) into a pair of length and offset.
Examples

<table>
<thead>
<tr>
<th>Number to convert</th>
<th>Unary code</th>
<th>Length</th>
<th>Offset</th>
<th>Gamma code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1110</td>
<td>10</td>
<td>1</td>
<td>10,1</td>
</tr>
<tr>
<td>24</td>
<td>1110</td>
<td>1000</td>
<td>1100,1000</td>
<td></td>
</tr>
</tbody>
</table>

Offset is the binary rep of the number but with the leading 1 removed. Length is the length of the offset in unary code. Is decoded by reading the unary code up to the 0 that terminates it. Now we know how long the offset is. The offset can then be read correctly and the 1 that was chopped off in encoded is appended.

**Simple9 (S9) coding**

Try to produce a word-aligned code. Basic unit: 32 bits. Try to pack several numbers into one word (32 bits) Each word is split into 4 control bits and 28 data bits. These 4 bits gives information of which number(s) that is stored in the other 28 bits.

**Universal code**

A code like gamma code with the property of being a within a factor of optimal for an arbitrary distribution P is called universal.

**Prefix free**

Gamma codes, for instance, is prefix free because no gamma code is the prefix of another. This means that there is always a unique decoding of a sequence of gamma codes and we do not need delimiters between them.

**Parameter free**

Gamma codes, for instance, is parameters free. With coding that contains parameters, these must be stored and retrieved. And in dynamic indexing, the distribution of gaps can change, so that the original parameters are no longer appropriate.

**Rice coding**

Take average of all gaps in PLs. Round this to smaller power of two. If average is 113. \(2^6 = 64\). Therefore, \(k = 6\).
Example
Encode the 8-bit value 18 (0b00010010) when K = 4 (M = 16)

1. $S \& (M - 1) = 18 \& (16 - 1) = 0b00010010 \& 0b1111 = 0b0010$
2. $S \gg K = 18 \gg 4 = 0b00010010 \gg 4 = 0b0001$ (10 in unary)

So the encoded value is 10 0010, saving 2 bits.

Example
Encode 33 when K = 6. M = $2^6 = 64$

1. $33 \& (63) = 100001 \& 111111 = 100001$
2. $33 \gg 6 = 100001 \gg 6 = 0$. (0 in unary)

Therefore rice coding of 33 is 0 100001

Example
Encode 161 when K = 6. M = $2^6 = 64$

1. $161 \& 63 = 10100001 \& 111111 = 100001$
2. $161 \gg 6 = 10100001 \gg 6 = 10$ (110 in unary)

Therefore rice coding of 161 is 110 100001

Better compression than var-byte, but slightly slower.

There are several more that isn’t covered

Lecture 6 Scoring, term weighting, and vector space model

Metadata
Digital docs generally encode, in machine-recongnizable form, certain metadata associated with each document. Such as author(s), title, and date of publication.

Field
Examples of fields: Date of creation, format, author, title.
Parametric indexes

There is one parametric index for each field. It allows us to select only the docs matching a data specified in the query.

Zones

Similar to fields, except the content of a zone can be arbitrary free text.

Weighted zone scoring

Assigns to the pair (query, doc) a score in the interval $[0, 1]$, by computing a linear combination of zone scores.

Term frequency

Number of occurrences of a term in a doc.

Bag of words

A doc representation where the exact ordering of the terms in a document is ignored. We only retain information on the number of occurrences of each term. Therefore, “Mary is quicker than John” gives the same document representation as “John is quicker than Mary”.

Document frequency

Defined to be the number of docs in the collection that contains a term $t$.

Inverse document frequency

$idf = \log (N/df)$. Thus, the idf of a rare term is high.

Tf-idf

$tf-idf = tf \times idf$

Properties

1. Highest when a term occurs many times within a small number of docs
2. Lower when the term occurs fewer times in a document, or occurs in many documents

3. Lowest when the term occurs in virtually all documents.

**Document vector**

Can view each document as a vector with one component corresponding to each term in the dictionary, together with a weight for each component that is given by tf-idf.

**Vector space model**

The representation of a set of docs as vectors in a common vector space is known as the vector space model.

**Cosine similarity**

The standard way of quantifying the similarity between two documents or a query and a document is to compute the cosine similarity where the numerator represents the dot product and the denominator is the product of their Euclidean length. This measure is the cosine of the angle $\Theta$ between two vectors.

**Term-document matrix**

Viewing a collection of N docs as a collection of vectors leads to a natural view of a collection as term-document matrix.

**Term-at-a-time**

The process of adding contributions one query term at a time is known as term-at-a-time scoring or accumulation, and the N elements of the array “Scores” are therefore known as accumulators.

**Document-at-a-time**

In a postings traversal, if we compute the scores of one document at a time, its called document-at-a-time scoring. Basically, if you use the merging/intersection algorithm from chapter 1, you are using document-at-a-time scoring. More info [here](#)
Variant tf-idf functions

Sublinear tf-scaling

Use the logarithm of the term frequency, because a term that occurs twenty times is (often) not 20 times more important than a term that occurs once. Therefore: \( \text{tf-idf} = \text{wf} \times \text{idf} \) where \( \text{wf} = 1 + \log \text{tf} \) if \( \text{tf} > 0 \) \( \text{wf} = 0 \) otherwise

Maximum tf normalization

Normalize the tf weights of all terms occurring in a document by the maximum tf in that doc.

Lecture 7 Computing scores in a complete search system

Given the cosine similarity from each doc vector \( V(d) \) to the query vector \( V(q) \), we want the top \( K \) highest scoring docs and present it to the user. This chapter will now consider schemes by which we produce \( K \) docs that are likely to be among the \( K \) highest scoring docs for a given query. In doing so, we want to lower the cost of computing the \( K \) docs. The following procedures have the following two-step scheme:

1. Find a set of \( A \) docs that are contenders where \( K < |A| \ll N \). \( A \) does not necessarily contain the \( K \) top-scoring docs for the query, but is likely to have many docs with scores near those of top \( K \).

2. Return the \( K \) top-scoring docs in \( A \).

Now, 5 schemes for doing so will be presented:

Index elimination

1. We only consider docs containing terms whose idf exceeds a preset threshold. Thus, in the postings traversal, we only traverse the postings for terms with high idf. Additional benefit: The postings lists of low-idf terms are generally long.

2. We only consider docs that contain many of the query terms.
Champion lists/fancy lists/top docs

Precompute, for each term $t$ in the dictionary, the set of the $r$ docs with the highest weights for $t$; the value of $r$ is chosen in advance. We call the set of $r$ docs the champion list for term $t$. Now, given a query $q$ we create a set $A$ as follows: We take the union of the champion lists for each of the terms comprising $q$. We now restrict cosine computation to only the docs in $A$. Important to find a good value for $r$.

Static quality scores and ordering

Further development of champion lists. Several ideas:

1. Incorporate a static quality score that is query independent. Then the score for a doc can be some combination of the quality score together with a tf-idf score. The precise combination may be determined by machine learning.

2. One can also consider ordering the postings by the static quality scores. One can still use the normal PL intersection algorithm, as long as there are a single common ordering.

3. Then, one can also maintain for each term $t$ a global champion list of the $r$ docs with the highest values for $g(d) + \text{tf-idf}$. So at query time, we only compute the net-scores for docs in the union of these global champion lists.

4. We maintain for each term $t$ two PLs consisting of disjoint sets of docs, each sorted by the static quality score. The first list, called high, contains the $m$ docs with the highest tf values for $t$. The second list, called low, contains all other docs containing $t$. When we process a query, we first scan only the high lists, if we don’t obtain $K$ docs, we continue with scanning the low lists.

Impact ordering

PLs are not ordered by a single common ordering. Instead, order them in decreasing order of tf. To ideas:

1. When traversing a PL, we stop after considering a prefix of the PL, either after a fixed number or if the tf has dropped below some threshold.
2. When accumulating scores in the outer loop, we consider the query terms in decreasing order of idf. If the idf is very low, and changes are minimal, we simply terminate the accumulation.

Cluster pruning
Incorporate a preprocessing step during which we cluster the doc vectors. Then, at query time, we consider only docs in a small number of clusters as candidates for which we compute cosine scores. Use of randomly chosen leaders is fast and likely to reflect the distribution of the doc vectors in the vector space.

Now, we combine the ideas developed to describe a search system that retrieves and scores docs.

Tiered indexes
What if the set of A contenders are fewer than the K docs? One can use tiered indexes. If we fail to get K results from tier 1, query processing falls back to tier 2 and so on.

Query term proximity
Let $\omega$ be the width of the smallest window in a doc $d$ that contains all the query terms. The smaller the $\omega$ is, the better the doc matches the query.

Lecture 8 Evaluation in information retrieval

Gold standard/ground truth
The decision of whether a document is either relevant or non-relevant.

Development test collection
Used to evaluate the IR system. Many systems contain various weights (parameters) that can be used to tune system performance. But one cannot tune these parameters to maximize performance on that collection. Therefore, one must have a development test collection. (As in machine learning, where you have a test set that is different from the training set).
**Precision**

Precision = number of relevant retrieved items / number of retrieved items

**Recall**

Recall = number of relevant items retrieved / number of relevant items

**Accuracy**

<table>
<thead>
<tr>
<th>Retrieved</th>
<th>Relevant</th>
<th>Non-relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True positives</td>
<td>False positives</td>
</tr>
<tr>
<td>Not-retrieved</td>
<td>False negatives</td>
<td>True negatives</td>
</tr>
</tbody>
</table>

Accuracy = \(\frac{tp + tn}{tp + fp + fn + tn}\) Why accuracy is not a good measure for IR problems:

- Normally, non-relevant documents consists of 99.9% of the collection. To maximize the accuracy one can deem all docs as non-relevant.

Great to have to numbers to evaluate a IR system (precision and recall) because one is often more important than the other.

**F measure**

Precision and recall are often a tradeoff: Recall is a nondecreasing function. On the other hand, precision usually decreases as the number of retrieved docs is increased. In general, we want to get some amount of recall while tolerating only a certain percentage of false positives. See [wikipedia](https://en.wikipedia.org) for formula. Values for \(\beta < 1\) emphasize precision, but \(\beta > 1\) emphasize recall. One cannot use the arithmetic mean because one can always get 100% recall by returning all the docs, and therfore always get a minimum of 50% arithmetic mean.

**Precision-recall curve**

For each set of retrieved documents one can plot the precision recall curve. They have a distinct sawtooth shape. If the \((k + 1)\)th doc retrieved is non-relevant, then recall is the same, but precision has dropped. If it is relevant, then both precision and recall increase, and the curve jags up and to the right.
Interpolated precision

To remove the jiggles in a precision-recall curve.

Mean average precision

For one query: Take the average of precision value obtained for the top K docs, each time a relevant doc is retrieved. For all the queries, take the average of the values obtained with the above scheme, then this is the mean average precision. Therefore, each query counts equally.

Precision at K

Precision at k documents (P@k) is still a useful metric (e.g. P@10 corresponds to the number of relevant results on the first search results page), but fails to take into account the positions of the relevant documents among the top k.

R precision

To cope with the precision at K docs, is R-precision. It requires having a set of known relevant docs Rel, from which we calculate the precision of the top Rel documents returned.

Pooling

Given info needs and docs, you need to collect relevance assessments (overslag). This is a time-consuming and expensive process involving human beings. For a large collection, it is usual for relevance to be assessed only for a subset of the docs for each query. The most standard approach is pooling, where relevance is assessed over a subset of the collection that is formed from the top k docs returned by a number of different IR system.

Kappa statistic

A measure of how much agreement between judges there is on relevance judgments. It is designed for categorical judgments and corrects a simple agreement rate for the rate of chance agreement.
Lecture 9 [TOMT]

Lecture 10 XML Retrieval

Structured retrieval
Search over structured documents

XML

XML element
A node in a tree

XML attribute
An element can have one or more XML attributes

XML DOM
Document Object Model. The standard way for accessing and processing XML docs. The DOM represents elements, attributes, and text within elements as nodes in a tree. With a DOM API, we can process a XML document by starting at the root element and then descending down the tree from parents to children.

XML Context/contexts
An XML path.

Schema
A schema puts constraints on the structure of allowable XML documents for a particular application. A schema for Shakespeare’s plays may stipulate that scenes can only occur as children of acts and that only acts and scenes have the number attribute.
NEXI
A common format of a XML query.

Structured document retrieval principle
The principle is as follows: A system should always retrieve the most specific part of a document answering the query.

Indexing unit
Which parts of a document to index.

Schema heterogeneity/diversity
In many cases, several different XML schemas occur in a collection because the XML documents in an IR application often come from more than one source. This is called schema heterogeneity. It presents yet another challenge(s):

1. Comparable elements may have different names like author vs creator.

2. The structural organization of the schemas may be different: Author names are direct descendants of the node autor, but in a different structure there can be firstname and lastname as direct children from author.

Extended query
We can support the user by interpreting all parent-child relationships in queries as descendant relationships with any number of intervening nodes allowed. These are extended queries.

Structural term
To index all paths that end in a single vocabulary term, in other words, all XML-context/term pairs. We call such an XML-context/term pair a structural term.

Text-centric XML
Where we match the text of the query with the text of the XML documents
Data-centric XML

Mainly encodes numerical and nontext attribute-value data. When querying data-centric XML, we want to impose exact match conditions in most cases. A query can be: “Find employees whose salary is the same this month as it was 12 months ago.” This query requires no ranking. It is purely structural and an exact matching of the salaries in the two time periods is probably sufficient to meet the users information need.

Lecture 11 Probabilistic information retrieval

Given the query, an IR system has an uncertain understanding of the information need. Given the query and a doc representation, a system has an uncertain guess of whether a doc has content relevant to the information need. Probability theory provides a principled foundation for such reasoning under uncertainty. This chapter provides one answer as to how to exploit this foundation to estimate how likely it is that a document is relevant to an information need.

Probability ranking principle (RPR)

The principle: If a reference retrieval system’s response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data.

1/0 loss

If you lose a point for either returning a nonrelevant doc or failing to return a relevant document, its called 1/0 loss. The goal is to return the best possible results as the top k docs. The RPR then says to simply rank all docs in decreasing order of \( P(R=1|d,q) \)

Bayes optimal rule

d is relevant if \( P(R=1|d, q) > P(R=0|d,q) \).
Binary independence model

Model to be used with the RPR. Here, binary is equivalent to Boolean. Docs and queries are both represented as binary term incidence vectors. That is, a doc $d$ is represented by the vector $x = (x_1, \ldots, x_m)$ where $x_t = 1$ if term $t$ is present or $x_t = 0$ if not. Many doc vectors can therefore be equivalent. The same goes for query vector. This assumption is far from correct, but it nevertheless often gives satisfactory results in practice.

Naive Bayes assumption

The presence or absence of a word in a doc is independent of the presence or absence of any other word. Makes it easier to estimate the probability of an entire term incidence vector occurring.

Retrieval status value

A logarithmic ranking model.

Relative frequency

A way to estimate the probability of an event from data where one counts the number of an event occurred divided by the total number of trials.

BMI25 weights/Okapi weighting

A probabilistic model sensitive to term frequency and doc length.

Lecture 12 [TOMT]

Lecture 13 Text classification and Naive Bayes

Standing query

Is like any other query except that is is periodically executed on a collection to which new docs are incrementally added over time.

Classification

Given a set of classes, we seeks to determine which class(es) a given object belongs to.
Routing/filtering
Classification using standing queries.

Topics
A general class is usually referred to as topic. For instance “China” or “coffee”

Applications of classification in IR

Preprocessing steps
Finding a doc’s encoding, truecasing, and identifying the language of a doc

Spam
Automatic detection of spam pages, which are then not included in the search engine index

Porn mode
Filter out

Sentiment detection
The automatic classification of a movie or product review as positive or negative.

Personal email sorting
Finding the correct folder for a new email.

Topic-specific or vertical search
Vertical search engines restrict searches to a particular topic. For example, the query “computer science” on a vertical search engine for the topic China will return a list of Chinese computer science departments with higher precision and recall than the query “computer science China” on a general purpose search engine.
Ranking function

The ranking function in ad hoc IR can also be based on a document classifier. More specified later. (sec. 15.4)

Rules in text classification (TC)

A rule captures a certain combination of keywords that indicates a class. Hand-coded rules have good scaling properties, but creating and maintaining them over time is labor intensive. In machine learning these rules are learned automatically from training data.

Statistical text classification

The approach where rules are learning automatically with machine learning.

Labeling

The process of annotating each doc with its class.

Document space

In TC we are given a description $d \in X$ of a document where $X$ is the document space. Typically, the document space is some type of high-dimensional space.

Space class

In TC, we are given a fixed set of classes $C = \{c_1, c_2, \ldots, c_j\}$ Typically, the classes are human defined for the needs of an application.

Training set

In TC, we are usually given a training set $\mathcal{D}$ of labeled docs $\langle d, c \rangle$ where $\langle d, c \rangle \in X \times C$

Learning method classifier

Using a learning method or learning algorithm, we then with to learn a classifier or classification function $\gamma$ that maps documents to classes: $\gamma: X \rightarrow C$
Surprised learning

The above type over learning is called supervised learning because a supervisor serves as a teacher directing the learning process. We denote the supervised learning method $\Gamma$ and write $\Gamma(D) = \gamma$. The learning method $\Gamma$ takes the training set $D$ as input and return the learning classification function $\gamma$.

Test setdata

Once we have learning $\gamma$, we can apply it to the test set, for example, the new doc, “first rpivate Chinese airline” whose class is unknown. The classification function hopefylle assigns the new document to class $\gamma(d') = \text{China}$.

Sparseness

The training data are never large enough to represent the frequency of rare events adequately. Therefore, the probability will often be zero.

Bernoulli model

Equivalent to the binary independence model, which generates an indicator for each term of the vocabulary, either 1 indicating presence of the term in the doc or 0 indicating absence.

Concept drift

The grandal change over time of the concept underlying a class like US president from George W. Bush to Barack Obama. The Bernoulli model is particularly robust with respect to this because the most important indicators of a class are less likely to change.

Feature selection

Is the process of selecting a subset of the terms occurring the training set and using only this subset as features in TC. Two main purposes:

1. It makes training and applying a classifier more efficient by decreasing the size of the vocabulary. Important for classifiers that are expensive to train (unlike NB)
2. It often increases classification accuracy by elimination noise features
Noise feature
Is one that, when added to the doc representation, increases the classification error on new data.

Example
Suppose a rare term, say arachnocentric, has no info about a class, say China, but all instances of arachnocentric happen to occur in China docs in our training set. Then the learning method might produce a classifier that misassigns test docs containing arachnocentric to China.

Overfitting
Such an incorrect generalization from the example above from an accidental property of the training set is called overfitting.

Mutual information (MI)
MI measures how much info the presence/absence of a term contributes to making the correct classification decision.

X² feature selection
In statistics, the X² test is applied to test the independence of two events. In feature selection, the two events are occurrence of the term and occurrence of the class. A high X² value indicates that the hypothesis of independence, which implies that expected and observed counts are similar, are incorrect.

Frequency-based feature selection
A feature selection method. It selects some frequent terms that have no specific info about the class.

Greedy feature selection
All the tree feature selection methods described (MI, X² and frequency based feature selection) are examples of greedy methods. They may select features that contribute no incremental information over previously selected features.
Two-class classifier

An approach to an any-of problem. You must learn several two-class classifier, one for each class, where the two-class classifier for class $c$ is the classifier for the two classes $c$ and its complement $\bar{c}$.

Effectiveness

Is a generic term for measures that evaluate the quality of classification decisions, including precision, recall, $F_1$ and accuracy.

Performance

Refers to the computational efficiency of classification and IR systems.

Macro/Micro- averaging

We often want to compute a single aggregate measure that combines the measures for individual classifier. There are two methods for doing this.

Macroaveraging

Computes a simple average over classes.

Microaveraging

Pools per-doc decision across classes, and then computes an effectiveness measure on the pooled contingency table.

The differences between the two methods can be large. Macroaveraging gives equal weights to each class, whereas microaveraging gives equal weight to each per-doc classification decision.

Development set

A set for testing while you develop your methods.

Lecture 14 Vector space classification

Goal: To develop a different representation for TC, the vector space model from chapter 6. Different from the rep used in last chapter where the doc rep in Naive Bayes was a sequence of terms or a binary vector.
**Contiguity hypothesis**
Document in the same class form a contiguous region and regions of different classes do not overlap.

**Prototype**
A centroid in the Rocchio classifier.

**Decision boundary**
Boundaries to separate the classes. To classify a new document, we determine the region it occurs in and assign it the class of that region.

**Multimodal class**
A class with several logical clusters.

**$k$ nearest neighbour/kNN classification**
A classifier who determines the decision boundary locally. For 1NN we assign each doc to the class of its closest neighbour. For kNN we assign each doc to the majority class of its $k$ closest neighbours where $k$ is a parameter. The rationale of kNN classification is that, based on the contiguity hypothesis, we expect a test doc $d$ to have the same label as the training docs located in the local region surrounding $d$.

**Memory based learning-instance-based learning**
kNN is an example of this. We do not perform any estimation of parameters. kNN simply memorizes all examples in the training set and then compares the test doc to them.

**Bayes error rate**
A measure of the quality of a learning method. The average error rate of classifiers learned by it for a particular problem.
Bias-variance tradeoff

There is a tradeoff when selecting a classifier that produces good classifiers across sets (small variance) and that can learn classification problems with very difficult decision boundaries (small bias).

Bias

Bias is large if the learning method produces classifiers that are consistently wrong. Bias is small if:

1. The classifiers are consistently right
2. Different training sets cause errors on different documents
3. Different training sets cause positive and negative errors on the same docs, but that average out close to 0.

Linear method like Rocchio and NB have high bias for nonlinear problems. Nonlinear methods like kNN have a low bias, because the decision boundaries are variable, depending on the distribution of docs in the training set, learned decision boundaries can vary greatly.

Variance

Is the variation of the prediction of learned classifiers. Is large if different give rise to very different classifiers. It is small if the training set has a minor effect on the classification decision the classifier makes, be they correct or incorrect. Variance measures how inconsistent the decisions are, not whether they are correct or incorrect.

Lecture 15 Support vector machines (SVM) and machine learning on docs

Margin

The distance from the decision surface to the closest data point determines the margin of the classifier.

Support vectors

The points that are closest to the decision boundary.
Kernel trick
SVMs provide an easy and efficient way of doing a mapping to higher dimensional space, which is referred to as the kernel trick.

Lecture 19 20 21 Web search (Fra foiler og boka)

Basics of web search
Challenges compared to “traditional” IR

- Scale (billions of web pages)
- Heterogenous content
- Trust
- Web users different from “traditional” IR users
- Business aspects (sponsored search)

Types of web queries
1. Information[~50%]: General info on topic Ex: Italian cuisine, Britney Spears family life
2. Navigational[~20%]: Search specific entity Ex: University of Oslo in Norway
3. Transactional[~30%]: Want to do something Ex: Car rental from Gardemoen.

Web queries
- Precision often more important than recall
  - Especially precision on top results
  - Necessary to filter untrusted pages/spam
  - Need to consider other qualities than relevance (trustworthiness, recency of content)
  - Recall only matters if number of matches is very small
- Query language must be lightweight (mostly phrase queries)
The web graph

- We refer to the hyperlinks onto a page as in-links/indegree, and out-degree of a web page to be the number or links out of it.

- Strongly connected: If there are pairs of pages such that one cannot proceed from one page to of the pair to the other by following hyper-links, these are not strongly connected.

Spamdexing

Manipulation of web content to appear artificially high on search results for particular keywords.

- Common spamming techniques:
  - Keyword stuffing, invisible text
  - Cloaking: Server returns fake content to web crawlers
  - Doorways: dummy start page carefully crafted for keywords
  - Optimisation of metadata on the page

- Counter measures:
  - Exploit “quality signals” (from web and from users) to determine whether a webpage is trustworthy
  - Analysis of web graph to detect suspicious lingaes
  - Machine learning to classify spam
  - Editorial intervention (blacklists etc.)

Web crawling and indexing

Is the process by which we gather pages from web to index them and support a search engine. The objective of crawling is to quickly and efficiently gather as many useful web pages as possible. A crawler must be:

Robust

The web contains servers that create spider traps, which are generators of web pages that misled crawlers into getting stuck fetching an infinite number of pages in a particular domain.
Polite
Avoid flooding servers and only crawl allowed pages
A crawler should be

Distributed
Should have the ability to execute in a distributed fashion across multiple machines

Scalable
The architecture should permit scaling up the crawl rate by adding extra machines and bandwidth

Quality
Must be biased toward fetching useful pages.
The fetched page is parsed, to extract both the text and the links from the page. The extracted text is fed to a text indexer. The extracted URLs/links are then added to a URL frontier, which at all times consists of URLs whose corresponding pages have yet to be fetched by the crawler. This process may be viewed as traversing the web graph.

Crawling architecture
1. The URL frontier, containing URLs yet to be fetched in the current crawl
2. A DNS resolution module that determines the web server from which to fetch the page specified by a URL
3. A parings module that extracts the text and set of links from a fetched web page
4. A duplicate elimination module that determines whether an extracted link is already in the URL frontier or has recently been fetched.

A crawler must crawl webpages that are of high quality and/or are frequently updated more often.

- URL frontier
  The URL frontier have a front queue and back queue to take into the
consideration the polite and prioritizing effects. Its goals are to ensure that

1. Only one connection is open to any host
2. A waiting time for a few seconds occurs between successive requests to a host
3. High-priority pages are crawled preferentially

The two major modules are a set of F front queues and a set of B back queues. Both FIFO queues. The front queue implements the prioritization and back queue the politeness.

Web indexing

Two types of index partitioning:

1. Partitioning by terms: Index terms divided in subsets, and each subset is allocated to a node
   - Pro: Greater concurrency
   - Cons: Must exchange and merge long PLs across nodes

2. Partitioning by documents: Each node is responsible for a local index for a subset of all documents (query sent to each node and the result are merged back)
   - Pro: Often easier to distribute, more efficient I/O on PLs
   - Cons: More disk seeks, need to calculate global statistics separately

Link analysis

The study on link analysis builds on two intuitions:

1. The anchor text pointing to page B is a good description of page B
2. The hyperlink from A to B represents an endorsement of page B, by the creator of page A. Therefore, good nodes will tend to point to good nodes, and bad nodes to bad nodes.
PageRank

The most well-known algorithm for ranking the quality of webpages according to their link structure (used by Google). It assigns a numerical score between 0 and 1 to each page, known as its PageRank. Given a web-search query, the search engine will give each web-page a score where amongst others, PageRank is used.

Given a web-surfer, he can switch to a new node (webpage) in two ways:

1. By clicking on a hyperlink on a node to the new node

2. “Teleport” from A to B, for example by typing URL in a browser. All possible webpages are likely. Therefore, if N is the total number of nodes in the web-graph, the teleport operation takes to surfer to each node with probability 1/N.

In assigning a PageRank score to each node, we use the teleport operation in two ways:

1. When at a node with no out-links, the surfer invokes the teleport operation

2. At any node that has hyperlinks (outgoing pages) the surfer invokes the teleport operation with probability $0 < \alpha < 1$ and a standard random walk (click on one of the hyperlinks at random) with probability $1 - \alpha$ where $\alpha$ is a fixed parameter chosen in advance, typically 0.1

• Markov Chain

A random walk in a web-graph can be represented with a Markov Chain. It is an N x N transition probability matrix P. Every index should be a number between 0 and 1, and the sum of all rows should be 1. The entry $P_{ij}$ tells us the probability that the state at the next time-step is j, conditioned on the current state being i. Each entry $P_{ij}$ is known as a transition probability and depends only on the current state i (i = row, j = column). This is known as the Markov property.

• How to compute the matrix P

One state for each web-page, and each transition probability representing the probability of moving from one webpage to another. If there is a hyperlink from page i to page j, then $A_{ij} = 1$. Otherwise $A_{ij} = 0$. We can derive P from A. If a row of A has no 1s, then divide each element by 1/N. For all other rows, proceed as follows:
1. Divide each 1 in A by the number of 1s in its row.
2. Multiply the resulting matrix by 1 - \( \delta \)
3. Add \( \delta / N \) to every entry of resulting matrix, to obtain P.

- The PageRank computation
The probability distribution of the surfers position at any time is the probability vector \( \vec{x} \). The probability distribution at \( t = 1 \) (started at \( t = 0 \)) is given by the probability vector \( \vec{x}P \), at \( t = 2 \) by \( (\vec{x}P)P = \vec{x}P \) and so on.

The RageRank is query-independent (a static quality score). On the other hand, the relative ordering of pages should, intuitively, depend on the query being served. For this reason, search engines use static quality measures such as PageRank just one of many factors in scoring a web page.
PageRank example

The graph

\[
\begin{array}{cccccc}
 & 1 & 2 & 3 & 4 & 5 \\
1 & 0.1 & 0.35 & 0.1 & 0.1 & 0.35 \\
2 & 0.1 & 0.1 & 0.1 & 0.6 & 0.1 \\
3 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\
4 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 \\
5 & 0.35 & 0.1 & 0.1 & 0.35 & 0.1 \\
\end{array}
\]

The start distribution:
\[x_{t0} = [1, 0, 0, 0, 0]\]
(which means the surfer is in state A). The next state (at \(t=1\)) will be:
\[x_{t1} = x_{t0} P = [0.1, 0.35, 0.1, 0.1, 0.35]\]
The next state (at \(t=2\)) will be:
\[x_{t2} = x_{t1} P = [0.2, 0.25, 0.145, 0.12, 0.385, 0.145]\]
\[x_{\infty} = [0.19, 0.19, 0.149, 0.286, 0.19]^{\top}\]

Figure 1: PageRank example