INTRODUCTION
AND THE BOOLEAN MODEL

What is Information Retrieval

2

- Manning et al, 2008

> Information retrieval (IR) is finding material . . .
> of an unstructured nature . . .
> that satisfies an information need
> from within large collections . . . .

7/

What is Information Retrieval

3

- Manning et al, 2008

> Information retrieval (IR) is finding material . . .
> of an unstructured nature . . .
> that satisfies an information need
> from within large collections . . . .

Document Collections

4

Document Collections

5

IR in the 17th century: Samuel Pepys, the famous English diarist, subject-indexed his treasured 1000+ books library with key words.

7/

Document Collections

6

What we mean here by document collections

7

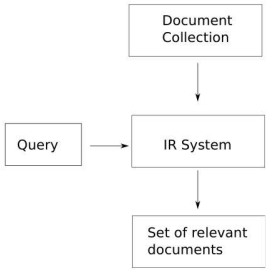
- Manning et al, 2008

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

- Document Collection: text units we have built an IR system over.
- Usually documents
- But could be
  - Memos , book chapters , Paragraphs , scenes of a movie , turns in a conversation...
- Lots of them

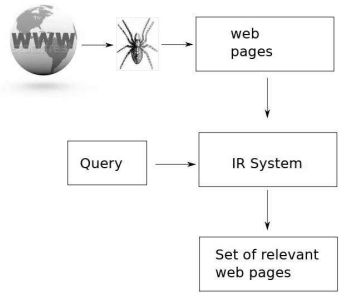
IR Basics

8



IR Basics

9



What is Information Retrieval

10

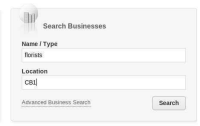
- Manning et al, 2008

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

Structured vs Unstructured Data

11

- Unstructured data means that a formal, semantically overt, easy-for-computer structure is missing.
  - In contrast to the rigidly structured data used in DB style searching (e.g. product inventories, personnel records)




```
SELECT *
FROM business_catalogue
WHERE category = 'florist'
AND city_zip = 'cb1'
```

- This does not mean that there is no structure in the data
  - Document structure (headings, paragraphs, lists. . .)
  - Explicit markup formatting (e.g. in HTML, XML. . .)
  - Linguistic structure (latent, hidden)

Information Needs and Relevance

12

- Manning et al, 2008

> 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).

- An information need is the topic about which the user desires to know more about.
- A query is what the user conveys to the computer in an attempt to communicate the information need.
- A document is relevant if the user perceives that it contains information of value with respect to their personal information need

7/

Types of information needs

13

- Manning et al, 2008

> Information retrieval (IR) is finding material . . . of an unstructured nature . . . that satisfies an information need from within large collections . . . .

- Known-item search
- Precise information seeking search
- Open-ended search ("topical search")

Information scarcity vs. information abundance

14

- Information scarcity problem (or needle-in-haystack problem): hard to find rare information
  - Lord Byron's first words ? Long sentence to the nurse in perfect English?
- Information abundance problem (for more clear-cut information needs) : redundancy of obvious information
  - What is toxoplasmosis?

Relevance

15

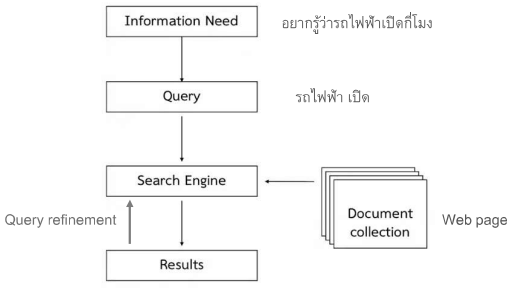
- Manning et al, 2008

> Information retrieval (IR) is finding material . . . of an unstructured nature . . . that satisfies an information need from within large collections . . . .

- Are the retrieved documents
  - about the target subject
  - up-to-date?
  - from a trusted source?
  - satisfying the user's needs?
- How should we rank documents in terms of these factors?

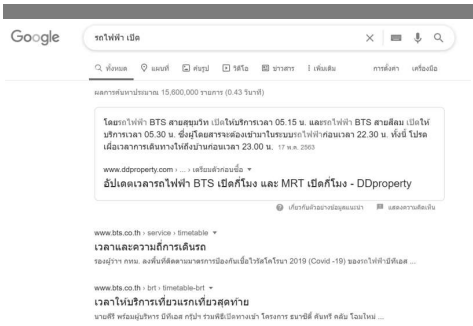
Classic Search Model

16



Challenges ?

17



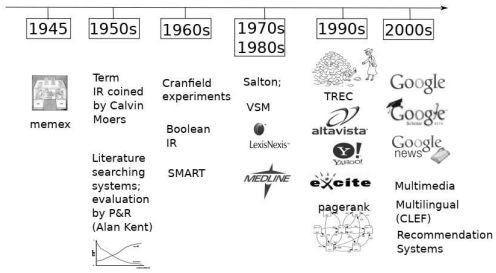
How well has the system performed?

18

- The effectiveness of an IR system (i.e., the quality of its search results) is determined by two key statistics about the system's returned results for a query:
  - 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?
- What is the best balance between the two?
  - Easy to get perfect recall: just retrieve everything
  - Easy to get good precision: retrieve only the most relevant

A short history of IR

19



IR today

20

- Web search (Google bing)
  - Search ground are billions of documents on millions of computers
  - issues: spidering; efficient indexing and search; malicious manipulation to boost search engine rankings
  - Link analysis
- Enterprise and institutional search (PubMed LexisNexis)
  - e.g company's documentation, patents, research articles
  - often domain-specific
  - Centralised storage; dedicated machines for search.
  - Most prevalent IR evaluation scenario: US intelligence analyst's searches
- Personal information retrieval (email, pers. documents;)
  - e.g., Mac OS X Spotlight; Windows' Instant Search
  - Issues: different file types; maintenance-free, lightweight to run in background

7/

IR for non-textual media

21

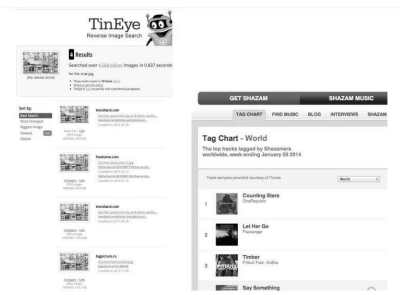
Areas of IR

23

- "Ad hoc" retrieval
- web retrieval
- Support for browsing and filtering document collections:
  - Clustering
  - Classification; using fixed labels
- Further processing a set of retrieved documents, e.g., by using natural language processing
  - Information extraction
  - Summarisation
  - Question answering

Similarity Searches

22



Boolean Retrieval

24

- In the Boolean retrieval model we can pose any query in the form of a Boolean expression of terms
- i.e., one in which terms are combined with the operators and, or, and not.
- Shakespeare example

Brutus AND Caesar AND NOT Calpurnia

23

- Which plays of Shakespeare contain the words Brutus and Caesar, but not Calpurnia?
- Naive solution: linear scan through all text – "grepping"
- In this case, works OK (Shakespeare's Collected works has less than 1M words).
- But in the general case, with much larger text collections, we need to index.
- Indexing is an offline operation that collects data about which words occur in a text, so that at search time you only have to access the precompiled index.

The term-document incidence matrix

26

- Main idea: record for each document whether it contains each word out of all the different words Shakespeare used (about 32K).

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

- Matrix element (t, d) is 1 if the play in column d contains the word in row t, 0 otherwise.

Query "Brutus AND Caesar AND Calpunia"

27

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

- Bitwise AND return one document, "Julius Caesar".

Query "Brutus AND Caesar AND NOT Calpunia"

28

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

7/

Query "Brutus AND Caesar AND NOT Calpunia"

29

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

Query "Brutus AND Caesar AND NOT Calpunia"

30

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

- Bitwise AND returns two documents, "Antony and Cleopatra" and "Hamlet".

The results: two documents

31

Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to Dominitus Enobarbus]: Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring, and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

32

Thank you

7/1

INVERTED INDEX

Boolean Retrieval

- In the Boolean retrieval model we can pose any query in the form of a Boolean expression of terms
- i.e., one in which terms are combined with the operators and, or, and not.
- Shakespeare example

Brutus AND Caesar AND NOT Calpurnia

- Which plays of Shakespeare contain the words Brutus and Caesar, but not Calpurnia?
- Naive solution: linear scan through all text – "grepping"
- In this case, works OK (Shakespeare's Collected works has less than 1M words).
- But in the general case, with much larger text collections, we need to index.
- Indexing is an offline operation that collects data about which words occur in a text, so that at search time you only have to access the precompiled index.

The term-document incidence matrix

- Main idea: record for each document whether it contains each word out of all the different words Shakespeare used (about 32K).

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

- Matrix element (t, d) is 1 if the play in column d contains the word in row t, 0 otherwise.

7/1
7/1

Query "Brutus AND Caesar AND Calpunia"

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

- Bitwise AND return one document, "Julius Caesar".

Query "Brutus AND Caesar AND NOT Calpunia"

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

Query "Brutus AND Caesar AND NOT Calpunia"

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

Query "Brutus AND Caesar AND NOT Calpunia"

- We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and Calpurnia :

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

- Bitwise AND returns two documents, "Antony and Cleopatra" and "Hamlet".

7/1

The results: two documents

9

Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to Dominitus Enobarbus]: Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring, and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Computer Storage

10

- Hard Drive

Disk

SSD

- RAM

Bigger collections

11

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

- Consider N=1M documents, each with about 1000 tokens
- 1000M tokens at avg 6 Bytes per token => 6GB
- Assume there are 500,000 distinct terms in the collection
- Size of incidence matrix is then 500,000 ×1M × 1 Byte => 500GB
- Half a trillion 0s and 1s

Can't build the Term-Document incidence matrix

12

- Observation: the term-document matrix is very sparse
- Contains no more than one billion 1s.
- Better representation: only represent the things that do occur
- Term-document matrix has other disadvantages, such as lack of support for more complex query operators (e.g., proximity search)
- We will move towards richer representations, beginning with the inverted index.

7/1
7/1

The inverted index

13

- The inverted index consists of
  - a dictionary of terms (also: lexicon, vocabulary)
  - and a postings list for each term, i.e., a list that records which documents the term occurs in.

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Caesar → 1 → 2 → 4 → 5 → 6 → 16 → 57 → 132 → 179

Calpurnia → 2 → 31 → 54 → 101

Processing Boolean Queries: conjunctive queries

14

Our Boolean Query

`Brutus AND Calpurnia`

Locate the postings lists of both query terms and intersect them.

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia → 2 → 31 → 54 → 101

Intersection 2 31

Note: this only works if postings lists are sorted

Algorithm for intersection of two postings

15

```
INTERSECT (p1, p2)
1   answer ← <>
2   while p1 ≠ NIL and p2 ≠ NIL
3   do if docID(p1) = docID(p2)
4      then ADD (answer, docID(p1))
5           p1 ← next(p1)
6           p2 ← next(p2)
7      if docID(p1) < docID(p2)
8         then p1← next(p1)
9         else p2← next(p2)
10  return answer
```

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia → 2 → 31 → 54 → 101

Intersection 2 31

Complexity of the Intersection Algorithm

16

- Bounded by worst-case length of postings lists
- Thus "officially" O(N), with N the number of documents in the document collection
- But in practice much, much better than linear scanning, which is asymptotically also O(N)

7/1

Query Optimisation: conjunctive terms

17

Organise order in which the postings lists are accessed so that least work needs to be done

Brutus AND Caesar AND Calpurnia

Process terms in increasing document frequency: execute as

(Calpurnia AND Brutus) AND Caesar

Brutus 8 → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Caesar 9 → 1 → 2 → 4 → 5 → 6 → 16 → 57 → 132 → 179

Calpurnia 4 → 2 → 31 → 54 → 101

Query Optimisation: disjunctive terms

18

(maddening OR crowd) AND (ignoble OR strife) AND (killed OR slain)

- Process the query in increasing order of the size of each disjunctive term
- Estimate this in turn (conservatively) by the sum of frequencies of its disjuncts

Boolean queries: Exact match

19

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using AND, OR and NOT to join query terms
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
  - Email, library catalog, macOS Spotlight

Example: WestLaw

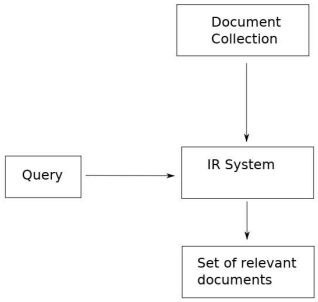
20

- http://www.westlaw.com/
- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992; new federated search added 2010)
- Tens of terabytes of data; ~700,000 users
- Majority of users still use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM /3 = within 3 words, /S = in same sentence

7/1
7/1

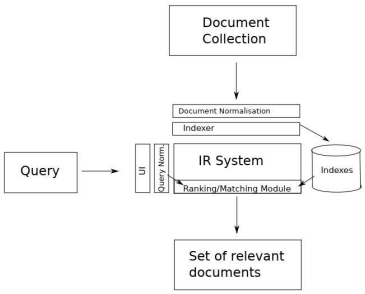
IR System Components

21



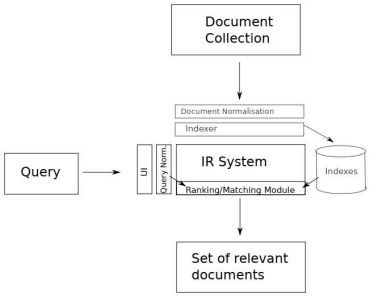
IR System Components

22



IR System Components

23



Index construction

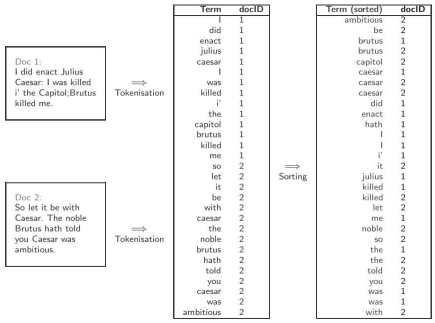
24

- The major steps in inverted index construction:
  - Collect the documents to be indexed.
  - Tokenize the text.
  - Perform linguistic preprocessing of tokens.
  - Index the documents that each term occurs in.

7/1

Example: index creation by sorting

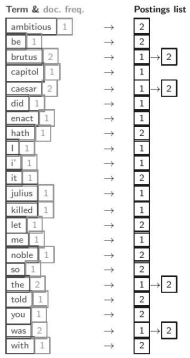
23



Index creation; grouping step

26

- Primary sort by term(dictionary)
- Secondary sort (within postings list) by documentID
- Document frequency(=length of postings list)
- keep Dictionary in memory
- keep Postings List (much larger) on disk



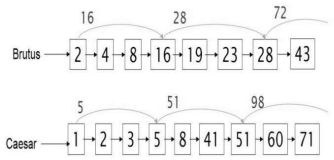
Data structures for Postings Lists

27

- Singly linked list
  - Allow cheap insertion of documents into postings lists (e.g., when recrawling)
  - Naturally extend to skip lists for faster access
- Variable length array
  - Better in terms of space requirements
  - Also better in terms of time requirements if memory caches are used, as they use contiguous memory
- Hybrid scheme: linked list of variable length array for each term.
  - write posting lists on disk as contiguous block without explicit pointers
  - minimises the size of postings lists and number of disk seeks

Optimisation: Skip Lists

28

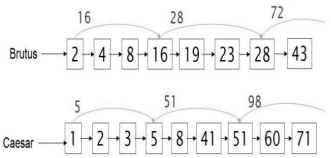


- Some postings lists can contain several million entries
- Check skip list if present to skip multiple entries
- sqrt(L) Skips can be placed evenly for a list of length L.

7/1
7/1

Tradeoff Skip Lists

29



- Number of items skipped vs. frequency that skip can be taken
- More skips: each pointer skips only a few items, but we can frequently use it.
- Fewer skips: each skip pointer skips many items, but we can not use it very often.
- Skip pointers used to help a lot, but with today's fast CPUs, they don't help that much anymore.

Algorithm: single-pass in-memory indexing or SPIMI

30

- As we build index, we parse docs one at a time.
- The final postings for any term are incomplete until the end.
- But for large collections, we cannot keep all postings in memory and then sort in-memory at the end
- We cannot sort very large sets of records on disk either (too many disk seeks, expensive)
- Thus: We need to store intermediate results on disk.
- We need a scalable Block-Based sorting algorithm.

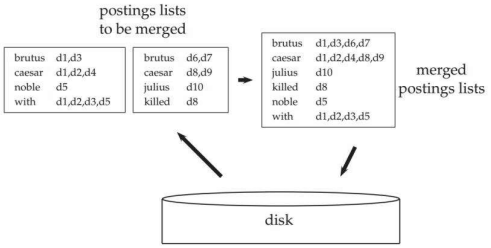
Single-pass in-memory indexing

31

- Abbreviation: SPIMI
- Key idea 1: Generate separate dictionaries for each block.
- Key idea 2: Accumulate postings in postings lists as they occur.
- With these two ideas we can generate a complete inverted index for each block.
- These separate indexes can then be merged into one big index.
- Worked example!

Single-pass in-memory indexing

32



Single-pass in-memory indexing

33

- We could save space in memory by assigning term-ids to terms for each block-based dictionary
- However, we then need to have an in-memory term-term-id mapping which often does not fit in memory (on a single machine at least)
- This approach is called blocked sort-based indexing BSBI

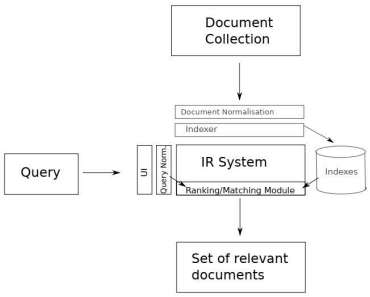
34

Thank you

DOCUMENT AND TERM NORMALIZATION

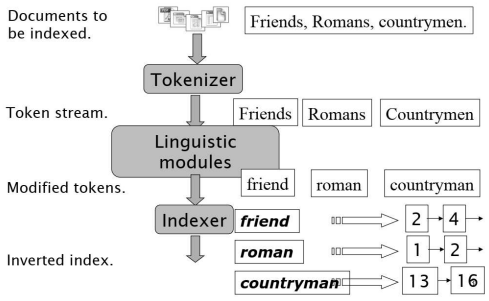
IR System Components

2



Overview

3



Document and Term Normalization

4

- To build an inverted index, we need to get from
  - Input: Friends, Romans, countrymen. So let it be with Caesar. . .
  - Output: friend roman countryman so
  - Each token is a candidate for a postings entry.
  - What are valid tokens to emit?

Documents

5

- Up to now, we assumed that
  - We know what a document is.
  - We can "machine-read" each document
- More complex in reality

Parsing a document

6

- We need do deal with format and language of each document
- Format could be excel, pdf, latex, word. . .
- What language is it in?
- What character set is it in?
- Each of these is a statistical classification problem
- Alternatively we can use heuristics

Character decoding

7

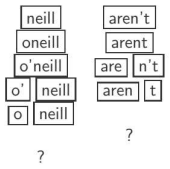
Text is not just a linear stream of logical "characters"...

- Determine correct character encoding (Unicode UTF-8) – by ML or by metadata or heuristics.
- Compressions, binary representation (DOC)
- Treat XML characters separately (`&`)

Tokenisation

8

Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.



Tokenisation problems: One word or two? (or several)

9

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

Chinese: No Whitespace

10

莎拉波娃现在居住在美国东南部的佛罗里达。今年 4 月 9 日，莎拉波娃在美国第一大城市纽约度过了 1 8 岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

- Need to perform word segmentation
- Use a lexicon or supervised machine-learning

Chinese: Ambiguous segmentation

11

和尚

- As one word, means "monk"
- As two words, means "and" and "still"

Japanese

12

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるMOTTAINAIキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

- Different scripts (alphabets) might be mixed in one language.
- Japanese has 4 scripts: kanja, katakana, hiragana, Romanji
- no spaces

Arabic script and bidirectionality

13

- Direction of writing changes in some scripts (writing systems); e.g., Arabic.

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.
← → ← → ← START
'Algeria achieved its independence in 1962 after 132 years of French occupation.'

- Rendering vs. conceptual order
- Bidirectionality is not a problem if Unicode encoding is chosen

Accents and diacritics

14

- résumé vs. resume
- Universität
- Meaning-changing in some languages:

  peña = cliff, pena = sorrow (Spanish)

- Main questions: will users apply it when querying?

Normalization

13

- Need to normalise words in the indexed text as well as query terms to the same form
- Example: We want to match U.S.A. to USA
- We most commonly implicitly define equivalence classes of terms.
- Alternatively, we could do asymmetric expansion:

window → window, windows
windows → Windows, windows, window
Windows → Windows

- Either at query time, or at index time
- More powerful, but less efficient

Numbers

16

20/3/91
3/20/91
Mar 20, 1991
B-52
100.2.86.144
(800) 234-2333
800.234.2333

- Older IR systems may not index numbers...
- ... but generally it's a useful feature.

Case Folding

17

- Reduce all letters to lower case
- Even though case can be semantically distinguishing

Fed vs. fed
March vs. march
Turkey vs. turkey
US vs. us

- Best to reduce to lowercase because users will use lowercase regardless of correct capitalisation.

Stop words

18

- Extremely common words which are of little value in helping select documents matching a user need

a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with

- Used to be standard in older IR systems.
- Need them to search for

to be or not to be
prince of Denmark
bamboo in water

- Length of practically used stoplists has shrunk over the years.
- Most web search engines do index stop words.

1/

More equivalence classing

19

- Thesauri: semantic equivalence, car = automobile
- Soundex: phonetic equivalence, Muller = Mueller;

Lemmatisation

20

- Reduce inflectional/variant forms to base form

am, are, is → be
car, car's, cars', cars → car
the boy's cars are different colours → the boy car be different color

- Lemmatisation implies doing "proper" reduction to dictionary headword form (the lemma)
- Inflectional morphology (cutting → cut)
- Derivational morphology (destruction → destroy)

Stemming

21

- Stemming is a crude heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatisation attempts to do with a lot of linguistic knowledge.
- language dependent, but fast and space-efficient
- does not require a stem dictionary, only a suffix dictionary
- Often both inflectional and derivational

automate, automation, automatic → automat

- Root changes (deceive/deception, resume/resumption) aren't dealt with, but these are rare

Porter Stemmer

22

- M. Porter, "An algorithm for suffix stripping", Program 14(3):130-137, 1980
- Most common algorithm for stemming English
- Results suggest it is at least as good as other stemmers
- Syllable-like shapes + 5 phases of reductions
- Of the rules in a compound command, select the top one and exit that compound (this rule will have affecte the longest suffix possible, due to the ordering of the rules).

Porter stemmer: selected rules

23

SSES → SS
IES → I
SS → S
S →

caresses → caress
cares → care

(*v*) ED →

plastered → plaster
bled → bled

(m>0) EED → EE

feed → feed
agreed → agree
BUT: freed, succeed

Three stemmers: a comparison

24

Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

Porter Stemmer

such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lovins Stemmer

such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Paice Stemmer

such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Linguistic Processing of Documents

25

From words to terms

- แปลงเอกสาร (e.g. email, PDF, word doc) ให้เป็น text file ธรรมดา
- Tokenize: *honey-roasted pork*
- Normalization: *U.S.A, USA —> usa ; naïve, naive —> naïve*

- Numbers : room 403 —> *room*
- Case folding: March —> *march*
- Stopwords: *the, a, to, of, in —> X*
- Lemmatisation : is am are —> *be*
- Stemming: *authorization, authorize, authorized —> authoriz*

Tokenize ภาษาไทย

26

ผลิตรายการ

- Word segmentation with Machine Learning :
  ผลิต, รายการ
- Dictionary :
  ผลิต, ราย, การ, ลิตร
- Character cluster:
  ผลิต, ราย, การ
- Character ngrams:
  ผล ลิ ติ ตร รา ยก กา ผลิ ลิตร ตรา ราย การ

Thai Character Cluster

27

<TCC> → ‘ก็’ | ‘อี’ | ‘หึ’
| <Cons> ‘รร’ ,<Cons> ‘์’,
| <Cons> <BCons> <Cons> ‘์’,
| <Cons> <TCC1> <Karan>
| <FSara><Cons> <TCC2> <Karan>

<TCC2> → <Cons>‘ะ’
| ‘ิ’<BCons>
| <USara>{<Tone>}<BCons> [‘า’|‘ะ’]
| {<Tone>}[‘า’|‘ะ’|‘ะ’]

<TCC1> → <DSara> {<Tone>}
| {<Tone>} ‘ำ’
| [‘ั’|‘ิ’] {<Tone>}<BCons>
| ‘ั’{<Tone>[‘ ุ’|‘ ู’]}
| ‘ิ’<BCons>
|<Tone>[<TSara>|<DSara>]{‘า’}<BCons>
| ‘ ู’{<Tone>{<BCons>}}
| ‘ิ’<Tone>
| {<Tone>} <Bsara>
| NULL

<Karan> → <Cons>{<Cons>}{[<DSara>|‘ ู’]}‘์’,
| NULL

Biword indexes

29

- Index every consecutive pair of terms in the text as a phrase.

Friends, Romans, Countrymen

Generates two biwords:
- friends romans
- romans countrymen

- Each of these biwords is now a vocabulary term.
- Two-word phrases can now easily be answered.

Phrase Queries

28

- We want to answer a query such as [cambridge university] – as a phrase.
- The Duke of Cambridge recently went for a term-long course to a famous university should not be a match
- About 10% of web queries are phrase queries.
- Consequence for inverted indexes: no longer sufficient to store docIDs in postings lists.
- Two ways of extending the inverted index:
  - biword index
  - positional index

Longer phrase queries

30

- A long phrase like cambridge university west campus can be represented as the Boolean query

cambridge university AND university west AND west campus

- We need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Issues with biword indexes

31

- Why are biword indexes rarely used?
- False positives, as noted above
- Index blowup due to very large term vocabulary

Positional indexes

32

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions (offsets)

Positional indexes: Example

33

Query: "to$_1$ be$_2$ or$_3$ not$_4$ to$_5$ be$_6$"

```
to,    993427:
< 1:   < 7, 18, 33, 72, 86, 231>;
  2:   <1, 17, 74, 222, 255>;
  4:   <8, 16, 190, 429, 433>;
  5:   <363, 367>;
  7:   <13, 23, 191>;
...    ...>

be,    178239:
< 1:   < 17, 25>;
  4:   < 17, 191, 291, 430, 434>;
  5:   <14, 19, 101>;
...    ...>
```

Document 4 is a match.
(As always: docid, term, doc freq; new: offsets)

Proximity search

34

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.

employment /4 place

- Find all documents that contain employment and place within 4 words of each other.
- HIT: Employment agencies that place healthcare workers are seeing growth.
- NO HIT: Employment agencies that have learned to adapt now place healthcare workers.

Proximity intersection

35

```
PositionalIntersect(p1, p2, k)
1 answer ←<>
2 while p1 6= nil and p2 6= nil
3 do if docID(p1) = docID(p2)
4      then l ← <>
5           pp1 ← positions(p1)
6           pp2 ← positions(p2)
7           while pp1 6= nil
8           do while pp2 6= nil
9              do if |pos(pp1)  pos(pp2)| ≤ k
10                  then Add(l , pos(pp2))
11                  else if pos(pp2) > pos(pp1)
12                     then break
13                  pp2 ← next(pp2)
14             while l ≠<> and |l [0]  pos(pp1)| > k
15             do Delete(l [0])
16             for each ps  l
17             do Add(answer , hdocID(p1), pos(pp1), psi)
18             pp1 ← next(pp1)
19          p1 ← next(p1)
20          p2 ← next(p2)
21      else if docID(p1) < docID(p2)
22           then p1 ← next(p1)
23           else p2 ← next(p2)
24 return answer
```

Combination scheme

36

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme. Faster than a positional index, at a cost of 26% more space for index.
- For web search engines, positional queries are much more expensive than regular Boolean queries.

RCV1 collection

37

- Shakespeare's collected works are not large enough to demonstrate scalable index construction algorithms.
- Instead, we will use the Reuters RCV1 collection.
- English newswire articles published in a 12 month period (1995/6)

N	documents	800,000
M	terms (= word types)	400,000
T	non-positional postings	100,000,000

Effect of preprocessing for Reuters

38

size of	word types (terms)			non-positional postings			positional postings (word tokens)		
	dictionary			non-positional index			positional index		
	size	Δ	cml	size	Δ	cml	size	Δ	cml
unfiltered	484,494			109,971,179			197,879,290		
no numbers	473,723	-2	-2	100,680,242	-8	-8	179,158,204	-9	-9
case folding	391,523	-17	-19	96,969,056	-3	-12	179,158,204	-0	-9
30 stopw's	391,493	-0	-19	83,390,443	-14	-24	121,857,825	-31	-38
150 stopw's	391,373	-0	-19	67,001,847	-30	-39	94,516,599	-47	-52
stemming	322,383	-17	-33	63,812,300	-4	-42	94,516,599	-0	-52

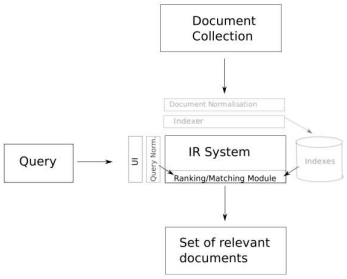
39

Thank you

INDEX REPRESENTATION AND TOLERANT RETRIEVAL

IR System components

2



Today: more indexing, some query normalisation

Upcoming

3

- Tolerant retrieval: What to do if there is no exact match between query term and document term
- Data structures for dictionaries
  - Hashes
  - Trees
  - k-term index
  - Permuterm index
- Spelling correction

4

Dictionaries

Inverted Index

5

Brutus 8 → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Caesar 9 → 1 → 2 → 4 → 5 → 6 → 16 → 57 → 132 → 179

Calpurnia 4 → 2 → 31 → 54 → 101

Dictionaries

6

- The dictionary is the data structure for storing the term vocabulary.
- Term vocabulary: the data
- Dictionary: the data structure for storing the term vocabulary
- For each term, we need to store a couple of items:
  - document frequency
  - pointer to postings list

How do we look up a query term q_i in the dictionary at query time?

7/2

Data structures for looking up terms

7

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
  - Is there a fixed number of terms or will it keep growing?
  - What are the relative frequencies with which various keys will be accessed?
  - How many terms are we likely to have?

Hashes

8

- Each vocabulary term is hashed into an integer, its row number in the array
- At query time: hash query term, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree. (Lookup time is constant.)
- Cons
  - no way to find minor variants (resume vs. résumé)
  - no prefix search (all terms starting with automat)
  - need to rehash everything periodically if vocabulary keeps growing

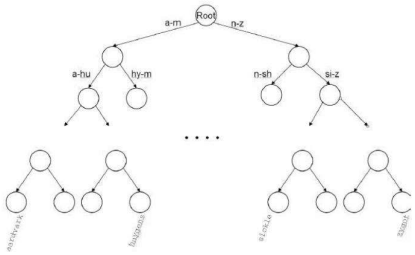
Trees

9

- Trees solve the prefix problem (find all terms starting with automat).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: O(logM), where M is the size of the vocabulary.
- O(logM) only holds for balanced trees.
- Rebalancing binary trees is expensive.
- B-trees mitigate the rebalancing problem.
- B-tree definition: every internal node has a number of children in the interval [a, b] where a, b are appropriate positive integers, e.g., [2, 4].

Binary tree

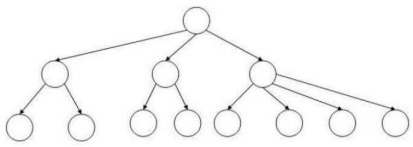
10



7/2
7/2

B-tree

11



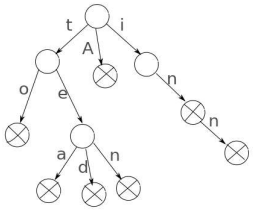
Trie

12

- An ordered tree data structure that is used to store an associative array
- The keys are strings
- The key associated with a node is inferred from the position of a node in the tree
  - Unlike in binary search trees, where keys are stored in nodes.
- Values are associated only with with leaves and some inner nodes that correspond to keys of interest (not all nodes).
- All descendants of a node have a common prefix of the string associated with that node → tries can be searched by prefixes
- The trie is sometimes called radix tree or prefix tree

Trie

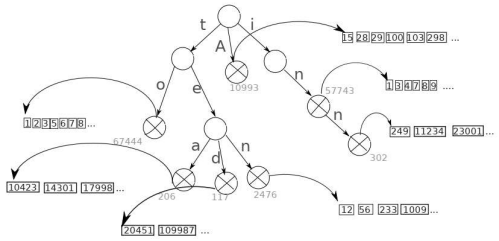
13



A trie for keys "A", "to", "tea", "ted", "ten", "in", and "inn".

Trie with postings

14



7/2

15 Wildcard queries

Wildcard queries

16

hel*

- Find all docs containing any term beginning with "hel"
- Easy with trie: follow letters h-e-l and then lookup every term you find there

*hel

- Find all docs containing any term ending with "hel"
- Maintain an additional trie for terms backwards
- Then retrieve all terms t in subtree rooted at l-e-h

In both cases:

- This procedure gives us a set of terms that are matches for wildcard query
- Then retrieve documents that contain any of these terms

How to handle * in the middle of a term

17

hel*o

- We could look up "hel*" and "*o" in the tries as before and intersect the two term sets.
  - Expensive
- Alternative: permuterm index
- Basic idea: Rotate every wildcard query, so that the * occurs at the end.
- Store each of these rotations in the dictionary (trie)

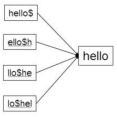
Permuterm index

18

For term hello: add

hello$, ello$h, llohe, lohel, o$hell, $hello

to the trie where $ is a special symbol



for hel*o, look up o$hel*

Problem: Permuterm more than quadrupels the size of the dictionary compared to normal trie (empirical number).

7/2

k-gram indexes

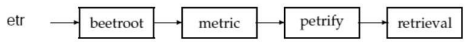
19

- More space-efficient than permuterm index
- Enumerate all character k-grams (sequence of k characters) occurring in a term

Bi-grams from April is the cruelest month

ap pr ri il l$ $i is s$ $t th he e$ $c cr ru ue el le es st t$ $m mo on nt th h$

- Maintain an inverted index from k-grams to the term that contain the k-gram



k-gram indexes

20

Note that we have two different kinds of inverted indexes:

- The term-document inverted index for finding documents based on a query consisting of terms
- The k-gram index for finding terms based on a query consisting of k-grams

Processing wildcard terms in a bigram index

21

- Query hel* can now be run as:

$h AND he AND el

- ... but this will show up many false positives like heel.
- Postfilter, then look up surviving terms in term–document inverted index.
- k-gram vs. permuterm index
  - k-gram index is more space-efficient
  - permuterm index does not require postfiltering.

22 Spelling correction

7/2

Spelling correction

23

an asterorid that fell form the sky

- In an IR system, spelling correction is only ever run on queries.
- The general philosophy in IR is: don't change the documents (exception: OCR'ed documents)
- Two different methods for spelling correction:
  - Isolated word spelling correction
    - Check each word on its own for misspelling
    - Will only attempt to catch first typo above
  - Context-sensitive spelling correction
    - Look at surrounding words
    - Should correct both typos above

Isolated word spelling correction

24

- There is a list of "correct" words – for instance a standard dictionary (Webster's, OED...)
- Then we need a way of computing the distance between a misspelled word and a correct word
  - for instance Edit/Levenshtein distance
  - k-gram overlap
- Return the "correct" word that has the smallest distance to the misspelled word.

informaton → information

Edit distance

25

- **Edit distance** between two strings s_1 and s_2 is the minimum number of basic operations that transform s_1 into s_2.
- **Levenshtein distance:** Admissible operations are insert, delete and replace

Levenshtein distance

dog	–	do	1 (delete)
cat	–	cart	1 (insert)
cat	–	cut	1 (replace)
cat	–	act	2 (delete+insert)

Levenshtein distance: Distance matrix

26

		s	n	o	w
	0	1	2	3	4
o	1	1	2	3	4
s	2	1	3	3	3
l	3	3	2	3	4
o	4	3	3	2	3

7/2
7/2

Edit Distance: Four cells

27

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** / **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** / **3**	3 **2** / 4 **2**	**2** 3 / 3 **2**	**3** 4 / **3** **3**	**4** **4** / **4** **4**
o	**4** / **4**	4 **3** / 5 **3**	**3** **3** / 4 **3**	**2** 4 / 4 **2**	4 5 / **3** **3**

Each cell of Levenshtein matrix

28

Cost of getting here from my upper left neighbour (by copy or replace)	Cost of getting here from my upper neighbour (by delete)
Cost of getting here from my left neighbour (by insert)	Minimum cost out of these

Dynamic Programming

29

Cormen et al:

- Optimal substructure: The optimal solution contains within it subsolutions, i.e, optimal solutions to subproblems
- Overlapping subsolutions: The subsolutions overlap and would be computed over and over again by a brute-force algorithm.

For edit distance:

- Subproblem: edit distance of two prefixes
- Overlap: most distances of prefixes are needed 3 times (when moving right, diagonally, down in the matrix)

Example: Edit Distance oslo – snow

30

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**				
s	**2** / **2**				
l	**3** / **3**				
o	**4** / **4**				

7/2

Example: Edit Distance oslo – snow

31

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	**2**	4	4	5
	4	5	**3**	4	**3**	4	**2**	**3**	3

Edit distance OSLO–SNOW is 3!

Example: Edit Distance oslo – snow

32

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	**2**	4	4	5
	4	5	**3**	4	**3**	4	**2**	**3**	3

cost	operation	input	output
1	delete	o	*
0	(copy)	s	s
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

Using edit distance for spelling correction

33

- Given a query, enumerate all character sequences within a preset edit distance
- Intersect this list with our list of "correct" words
- Suggest terms in the intersection to user.

k-gram indexes for spelling correction

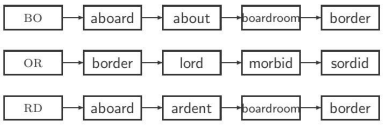
34

- Enumerate all k-grams in the query term

Misspelled word bordroom

bo – or – rd – dr – ro – oo – om

- Use k-gram index to retrieve "correct" words that match query term k-grams
- Threshold by number of matching k-grams
- Eg. only vocabularly terms that differ by at most 3 k-grams



7/2
7/2

Context-sensitive Spelling correction

35

One idea: hit-based spelling correction

flew form munich

- Retrieve correct terms close to each query term

flew → flea
form → from
munich → munch

- Holding all other terms fixed, try all possible phrase queries for each replacement candidate

flea form munich – 62 results
flew from munich –78900 results
flew form munch – 66 results

Not efficient. Better source of information: large corpus of queries, not documents

General issues in spelling correction

36

- User interface
  - automatic vs. suggested correction
  - "Did you mean" only works for one suggestion; what about multiple possible corrections?
  - Tradeoff: Simple UI vs. powerful UI
- Cost
  - Potentially very expensive
  - Avoid running on every query
  - Maybe just those that match few documents

Takeaway

37

- What to do if there is no exact match between query term and document term
- Datastructures for tolerant retrieval:
  - Dictionary as hash, B-tree or trie
  - k-gram index and permuterm for wildcards
  - k-gram index and edit-distance for spelling correction

38

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**								
s	**2**	**2**								
l	**3**	**3**								
o	**4**	**4**								

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	**1**	2						
		2								
s	**2**	**2**								
l	**3**	**3**								
o	**4**	**4**								

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	**1**	2						
		2	**1**							
s	**2**	**2**								
l	**3**	**3**								
o	**4**	**4**								

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	**1**	2	**2**	3				
		2	**1**		**2**					
s	**2**	**2**								
l	**3**	**3**								
o	**4**	**4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**				
s	**2** / **2**								
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4		
s	**2** / **2**								
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**		
s	**2** / **2**								
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5
s	**2** / **2**								
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**								
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2						
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**						
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	3				
l	**3** / **3**								
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2		
l	3 3				
o	4 4				

Example: Edit Distance OSLO – SNOW

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3	
l	3 3				
o	4 4				

Example: Edit Distance OSLO – SNOW

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	
l	3 3				
o	4 4				

Example: Edit Distance OSLO – SNOW

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4
l	3 3				
o	4 4				

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**								
	3								
o	**4**								
	4								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**						
	3	4							
o	**4**								
	4								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**						
	3	4	**2**						
o	**4**								
	4								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3				
	3	4	**2**	3					
o	**4**								
	4								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	3 / **2**	**3** / **3**	3 / **3**	**3** / 4	4 / **3**
l	**3** / **3**	3 / 4	**2** / **2**	**2** / 3	3 / **2**				
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	3 / **2**	**3** / **3**	3 / **3**	**3** / 4	4 / **3**
l	**3** / **3**	3 / 4	**2** / **2**	**2** / 3	3 / **2**	**3** / 3	4 /		
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	3 / **2**	**3** / **3**	3 / **3**	**3** / 4	4 / **3**
l	**3** / **3**	3 / 4	**2** / **2**	**2** / 3	3 / **2**	**3** / 3	4 / **3**		
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	3 / **2**	**3** / **3**	3 / **3**	**3** / 4	4 / **3**
l	**3** / **3**	3 / 4	**2** / **2**	**2** / 3	3 / **2**	**3** / 3	4 / **3**	**4** / **4**	
o	**4** / **4**								

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	3	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4	
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**	
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**	
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**	
o	**4**									
	4									

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	3	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4	
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**	
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**	
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**	
o	**4**	4	**3**							
	4	5								

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	3	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4	
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**	
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**	
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**	
o	**4**	4	**3**	**3**	**3**					
	4	5	**3**	4						

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	3	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4	
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**	
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**	
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**	
o	**4**	4	**3**	**3**	**3**					
	4	5	**3**	4						

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**				
	4	5	**3**	4	**3**				

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	**2**	4		
	4	5	**3**	4	**3**	4			

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	**2**	4		
	4	5	**3**	4	**3**	4	**2**		

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	1	**2**	2	**3**	3	**4**	4
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	**2**	4	4	5
	4	5	**3**	4	**3**	4	**2**	**3**	

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1** 2	2 **1**	**2** 3	3 **2**	**2** 4	3 **2**	4 **3**	5 **3**
s	**2**	**1** 2	3 **1**	**2** 3	**2** **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l	**3**	3 **2**	4 **2**	**2** 3	3 **2**	**3** 4	**3** **3**	**4** **4**	**4** **4**
o	**4**	4 **3**	5 **3**	**3** **3**	4 **3**	**2** 4	4 **2**	4 **3**	5 **3**

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1** 2	2 **1**	**2** 3	3 **2**	**2** 4	3 **2**	4 **3**	5 **3**
s	**2**	**1** 2	3 **1**	**2** 3	**2** **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l	**3**	3 **2**	4 **2**	**2** 3	3 **2**	**3** 4	**3** **3**	**4** **4**	**4** **4**
o	**4**	4 **3**	5 **3**	**3** **3**	4 **3**	**2** 4	4 **2**	4 **3**	5 **3**

Edit distance OSLO–SNOW is 3!

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1** 2	2 **1**	**2** 3	3 **2**	**2** 4	3 **2**	4 **3**	5 **3**
s	**2**	**1** 2	3 **1**	**2** 3	**2** **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l	**3**	3 **2**	4 **2**	**2** 3	3 **2**	**3** 4	**3** **3**	**4** **4**	**4** **4**
o	**4**	4 **3**	5 **3**	**3** **3**	4 **3**	**2** 4	4 **2**	4 **3**	5 **3**

How do I read out the editing operations that transform OSLO into SNOW?

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1** 2	2 **1**	**2** 3	3 **2**	**2** 4	3 **2**	4 **3**	5 **3**
s	**2**	**1** 2	3 **1**	**2** 3	**2** **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l	**3**	3 **2**	4 **2**	**2** 3	3 **2**	**3** 4	**3** **3**	**4** **4**	**4** **4**
o	**4**	4 **3**	5 **3**	**3** **3**	4 **3**	**2** 4	4 **2**	4 **3**	5 **3**

cost	operation	input	output

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	**2**	4	4	5
	4	5	**3**	4	**3**	4	**2**	**3**	**3**

cost	operation	input	output

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	2	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	2	4	4	5
	4	5	**3**	4	**3**	4	2	3	3

cost	operation	input	output

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
	2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	2	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	2	4	4	5
	4	5	**3**	4	**3**	4	2	3	3

cost	operation	input	output
0	(copy)	s	s

Example: Edit Distance OSLO – SNOW

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1**	**1**	2	**2**	3	**2**	4	4	5
	1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s	**2**	1	2	**2**	3	**3**	**3**	**3**	4
	2	3	1	**2**	**2**	**3**	**3**	4	**3**
l	**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
	3	4	**2**	3	2	**3**	**3**	**4**	**4**
o	**4**	4	**3**	**3**	**3**	2	4	4	5
	4	5	**3**	4	**3**	4	2	3	3

cost	operation	input	output
1	delete	o	*
0	(copy)	s	s

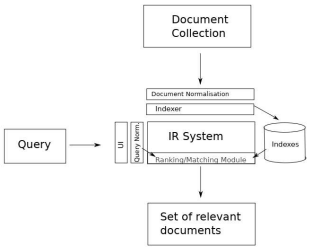
7/3

TERM WEIGHTING AND THE VECTOR SPACE MODEL

Upcoming

- Ranking search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- Term frequency: This is a key ingredient for ranking.
- Tf-idf ranking: best known traditional ranking scheme
- And one explanation for why it works: Zipf's Law
- Vector space model: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)

IR System components



Today: the matcher

4 Ranked retrieval

7/3
7/3

Ranked retrieval

- Thus far, our queries have been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and of the collection.
- Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users
- Don't want to write Boolean queries or wade through 1000s of results.
- This is particularly true of web search.

Problem with Boolean search: Feast or famine

- Boolean queries often have either too few or too many results.

Query 1
standard AND user AND dlink AND 650
→ 200,000 hits Feast!

Query 2
standard AND user AND dlink AND 650 AND no AND card AND found
→ 0 hits Famine!

- In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.
- In ranked retrieval, "feast or famine" is less of a problem.
- Condition: Results that are more relevant are ranked higher than results that are less relevant. (i.e., the ranking algorithm works.)

Scoring as the basis of ranked retrieval

- Rank documents in the collection according to how relevant they are to a query
- Assign a score to each query-document pair, say in $[0, 1]$.
- This score measures how well document and query "match".
- If the query consists of just one term ...

lioness

  - Score should be 0 if the query term does not occur in the document.
  - The more frequent the query term in the document, the higher the score
  - We will look at a number of alternatives for doing this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

  ($A \neq \emptyset$ or $B \neq \emptyset$)
- $\text{JACCARD}(A, A) = 1$
- $\text{JACCARD}(A, B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

7/3

Jaccard coefficient: Example

9

- What is the query-document match score that the Jaccard coefficient computes for:

Query
"ides of March"

Document
"Caesar died in March"

- $\text{JACCARD}(q, d) = 1/6$

What's wrong with Jaccard?

10

- It doesn't consider term frequency (how many occurrences a term has).
- It also does not consider that that some terms are inherently more informative than frequent terms.
- We need a more sophisticated way of normalizing for the length of a document.
  - Later in this lecture, we'll use $|A \cap B|/\sqrt{|A \cup B|}$ (cosine) ...
  - ... instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization.

11 Term frequency

Binary incidence matrix

12

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector $\in \{0, 1\}^{|V|}$.

7/3
7/3

Count matrix

13

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Term frequency tf

15

- The term frequency $\text{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We could just use tf as is ("raw term frequency").
- A document with $\text{tf} = 10$ occurrences of the term is more relevant than a document with $\text{tf} = 1$ occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Bag of words model

14

- We do not consider the order of words in a document.
- Represented the same way:

John is quicker than Mary
Mary is quicker than John

- This is called a bag of words model.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.

Instead of raw frequency: Log frequency weighting

16

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d} & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$\text{tf}_{t,d}$	0	1	2	10	1000
$w_{t,d}$	0	1	1.3	2	4

- Score for a document-query pair: sum over terms t in both q and d:

$$\text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document.

7/3

17

Zipf's Law and tf-idf weighting

Frequency in document vs. frequency in collection

18

- In addition, to term frequency (the frequency of the term in the document) ...
- ... we also want to reward terms which are rare in the document collection overall.
- Now: excursion to an important statistical observation about language.

Zipf's law

19

- How many frequent vs. infrequent terms should we expect in a collection?
- In natural language, there are a few very frequent terms and very many very rare terms.

Zipf's law

The i^{th} most frequent term has frequency cf_i proportional to $1/i$:

$$cf_i \propto \frac{1}{i}$$

- cf_i is collection frequency: the number of occurrences of the term t_i in the collection.

Zipf's law

20

Zipf's law

The i^{th} most frequent term has frequency cf_i proportional to $1/i$:

$$cf_i \propto \frac{1}{i}$$

- So if the most frequent term (*the*) occurs cf_1 times, then the second most frequent term (*of*) has half as many occurrences $cf_2 = \frac{1}{2}cf_1$ ...
- ... and the third most frequent term (*and*) has a third as many occurrences $cf_3 = \frac{1}{3}cf_1$ etc.
- Equivalent: $cf_i = p \cdot i^k$ and $\log cf_i = \log p + k \log i$ (for $k = -1$)

7/3
7/3

Zipf's Law: Examples from 5 Languages

21

Top 10 most frequent words in some large language samples:

	English		German		Spanish		Italian		Dutch	
1	the	61,847	der	7,377,879	que	32,894	non	25,757	de	4,770
2	of	29,391	die	7,036,092	de	32,116	di	22,868	en	2,709
3	and	26,817	und	4,813,169	no	29,897	che	22,738	het/'t	2,469
4	a	21,626	in	3,768,565	a	22,313	è	18,624	van	2,259
5	in	18,214	den	2,717,150	la	21,127	e	17,600	ik	1,999
6	to	16,284	von	2,250,642	el	18,112	la	16,404	te	1,935
7	it	10,875	zu	1,992,268	es	16,620	il	14,765	dat	1,875
8	is	9,982	das	1,983,589	y	15,743	un	14,460	die	1,807
9	to	9,343	mit	1,878,243	en	15,303	a	13,915	in	1,639
10	was	9,236	sich	1,680,106	lo	14,010	per	10,501	een	1,637
	BNC, 100Mw		"Deutscher Wortschatz", 500Mw		subtitles, 27.4Mw		subtitles, 5.6Mw		subtitles, 800Kw	

Zipf's law: Rank × Frequency ∼ Constant

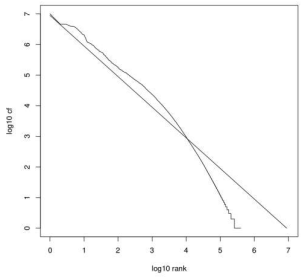
22

English:	Rank R	Word	Frequency f	$R \times f$
	10	he	877	8770
	20	but	410	8200
	30	be	294	8820
	800	friends	10	8000
	1000	family	8	8000

German:	Rank R	Word	Frequency f	$R \times f$
	10	sich	1,680,106	16,801,060
	100	immer	197,502	19,750,200
	500	Mio	36,116	18,059,500
	1,000	Medien	19,041	19,041,000
	5,000	Miete	3,755	19,041,000
	10,000	vorläufige	1.664	16,640,000

Zipf's law for Reuters

23



Fit is not great.

Desired weight for rare terms

24

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is rare in the collection (e.g., ARACHNOCENTRIC).
- A document containing this term is very likely to be relevant.
- → We want high weights for rare terms like ARACHNOCENTRIC.

7/3

Desired weight for frequent terms

23

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is frequent in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't ...
- ... but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → For frequent terms like GOOD, INCREASE, and LINE, we want positive weights ...
- ... but lower weights than for rare terms.

Document frequency

26

- We want high weights for rare terms like ARACHNOCENTRIC.
- We want low (positive) weights for frequent words like GOOD, INCREASE, and LINE.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is the number of documents in the collection that the term occurs in.

Effect of idf on ranking

29

- idf affects the ranking of documents for queries with at least two terms.
- For example, in the query "arachnocentric line", idf weighting increases the relative weight of ARACHNOCENTRIC and decreases the relative weight of LINE.
- idf has little effect on ranking for one-term queries.

Collection frequency vs. Document frequency

30

Term	Collection frequency	Document frequency
INSURANCE	10440	3997
TRY	10422	8760

- Collection frequency of t: number of tokens of t in the collection
- Document frequency of t: number of documents t occurs in
- Clearly, INSURANCE is a more discriminating search term and should get a higher weight.
- This example suggests that df (and idf) is better for weighting than cf (and "icf").

idf weight

27

- df_t is the document frequency, the number of documents that t occurs in.
- df_t is an inverse measure of the informativeness of term t.
- We define the idf weight of term t as follows:

$$\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$$

(N is the number of documents in the collection.)

- idf_t is a measure of the informativeness of the term.
- $\log \frac{N}{\text{df}_t}$ instead of $\frac{N}{\text{df}_t}$ to "dampen" the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

Examples for idf

28

Compute idf_t using the formula: $\text{idf}_t = \log_{10} \frac{1{,}000{,}000}{\text{df}_t}$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

7/3
7/3

tf-idf weighting

31

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

tf-idf weight

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$

- tf-weight
- idf-weight
- Best known weighting scheme in information retrieval
- Alternative names: tf.idf, tf x idf

32 The vector space model

7/3

Binary incidence matrix

33

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$.

Count matrix

34

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Binary → count → weight matrix

35

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

Documents as vectors

36

- Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.
- So we have a $|V|$-dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.

7/3
7/3

Queries as vectors

37

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
- proximity ≈ negative distance
- This allows us to rank relevant documents higher than nonrelevant documents

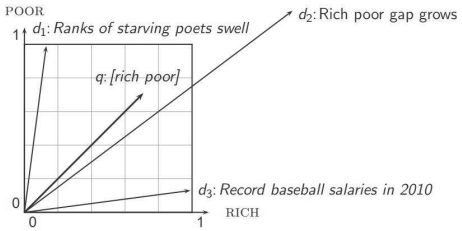
How do we formalize vector space similarity?

38

- First cut: (negative) distance between two points
- (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea ...
- ... because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

39



The Euclidean distance of $\vec{q}$ and $\vec{d}_2$ is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.

Cosine similarity between query and document

40

$$\cos(\vec{q},\vec{d}) = \text{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.
- This is the cosine similarity of $\vec{q}$ and $\vec{d}$ ..... or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

7/3

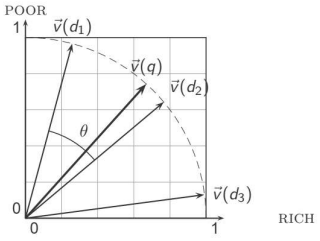
Cosine for normalized vectors

41

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.
- $\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$
  - (if $\vec{q}$ and $\vec{d}$ are length-normalized).

Cosine similarity illustrated

42



Cosine: Example

43

How similar are the following novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

Term frequencies (raw counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

Cosine: Example

44

	Term frequencies (raw counts)			Log frequency weighting			Log frequency weighting and cosine normalisation		
term	SaS	PaP	WH	SaS	PaP	WH	SaS	PaP	WH
AFFECTION	115	58	20	3.06	2.76	2.30	0.789	0.832	0.524
JEALOUS	10	7	11	2.0	1.85	2.04	0.515	0.555	0.465
GOSSIP	2	0	6	1.30	0.00	1.78	0.335	0.000	0.405
WUTHERING	0	0	38	0.00	0.00	2.58	0.000	0.000	0.588

- (To simplify this example, we don't do idf weighting.)
- cos(SaS,PaP) $\approx$ $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- cos(SaS,WH) ≈ 0.79
- cos(PaP,WH) ≈ 0.69

7/3
7/3

Components of tf-idf weighting

45

Term frequency		Document frequency		Normalization	
n (natural)	$\text{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\text{tf}_{t,d})$	t (idf)	$\log \frac{N}{\text{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \text{tf}_{t,d}}{\max_t(\text{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - \text{df}_t}{\text{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\text{tf}_{t,d})}{1 + \log(\text{ave}_{t \in d}(\text{tf}_{t,d}))}$				

Best known combination of weighting options

Default: no weighting

tf-idf example: lnc.ltn (ddd.qqq)

47

- N = 1,000,000

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

$\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$
$1/1.92 \approx 0.52$
$1.3/1.92 \approx 0.68$

Final similarity score between query and document: $\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$

tf-idf example

46

- We often use different weightings for queries and documents.
-
- Notation: ddd.qqq

Example: lnc.ltn

Document:
- l ogarithmic tf
- n o df weighting
- c osine normalization

Query:
- l ogarithmic tf
- t – means idf
- n o normalization

Summary: Ranked retrieval in the vector space model

48

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top K (e.g., $K = 10$) to the user


49