Lecture 1: Introduction and the Boolean Model

Information Retrieval
Computer Science Tripos Part II

Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

2017

[1] Adapted from Simone Teufel's original slides

Overview

1. Motivation
   - Definition of "Information Retrieval"
   - IR: beginnings to now
2. First Boolean Example
   - Term-Document Incidence matrix
   - The inverted index
   - Processing Boolean Queries
   - Practicalities of Boolean Search

What is Information Retrieval?

Manning et al, 2008:

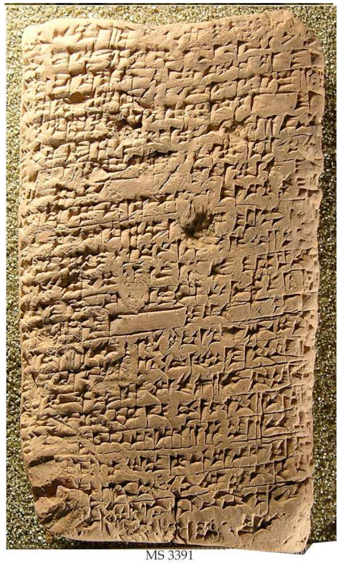
> Information retrieval (IR) is finding material . . . of an unstructured nature . . . that satisfies an information need from within large collections . . . .

What is Information Retrieval?

Manning et al, 2008:

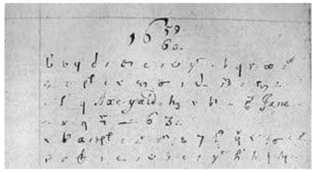
> Information retrieval (IR) is finding material . . . of an unstructured nature . . . that satisfies an information need from within large collections . . . .

Document Collections



MS 3391
Library catalogue. Babylonia, 2000-1600 BC

Document Collections



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

Document Collections

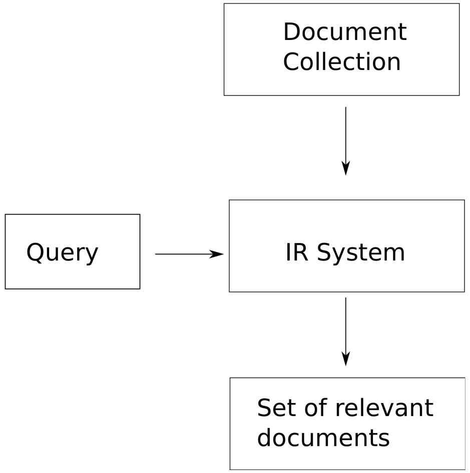
What we mean here by document collections

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



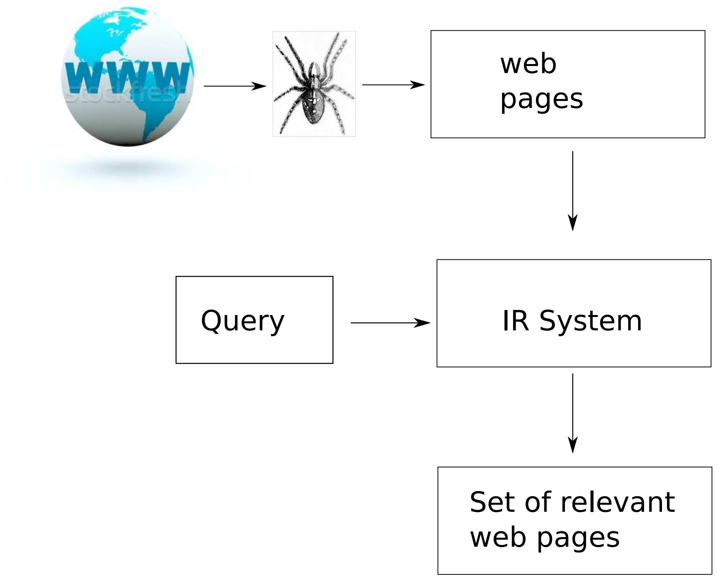
Document Collection

Query

IR System

Set of relevant documents

IR Basics



web pages

Query

IR System

Set of relevant web pages

What is Information Retrieval?

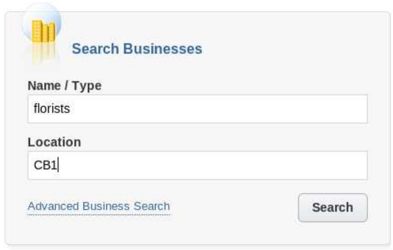
Manning et al, 2008:

> 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

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)



Name / Type: florists

Location: CB1

```
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

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.

Types of information needs

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

- Information scarcity problem (or needle-in-haystack problem): hard to find rare information
  - Lord Byron’s first words? 3 years old? Long sentence to the nurse in perfect English?

> . . . when a servant had spilled an urn of hot coffee over his legs, he replied to the distressed inquiries of the lady of the house, 'Thank you, madam, the agony is somewhat abated.' [not Lord Byron, but Lord Macaulay]

- Information abundance problem (for more clear-cut information needs): redundancy of obvious information
  - What is toxoplasmosis?

Relevance

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).

- 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?
- More on this in a lecture soon

How well has the system performed?

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:

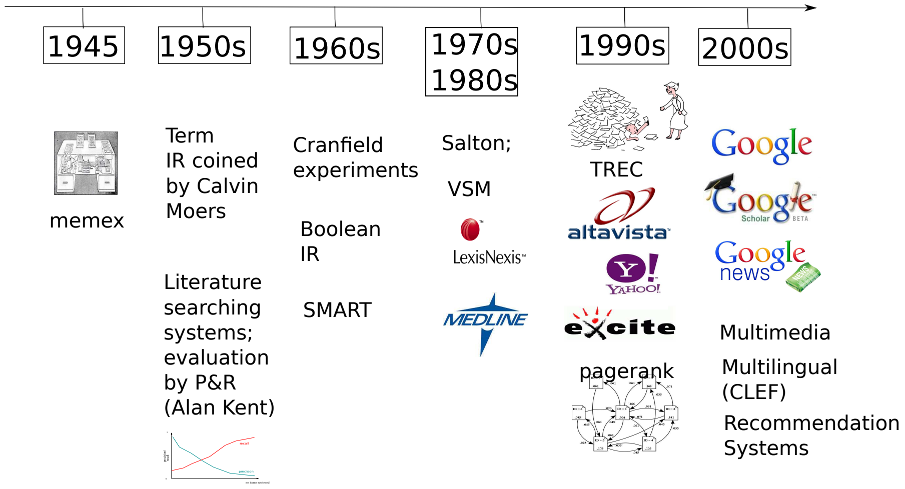
- Precision: What fraction of the returned results are relevant to the information need?
- Recall: What fraction of the relevant documents in the collection were returned by the system?
- What is the best balance between the two?
  - Easy to get perfect recall: just retrieve everything
  - Easy to get good precision: retrieve only the most relevant

There is much more to say about this – lecture 6

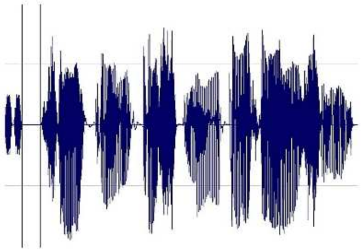
IR today

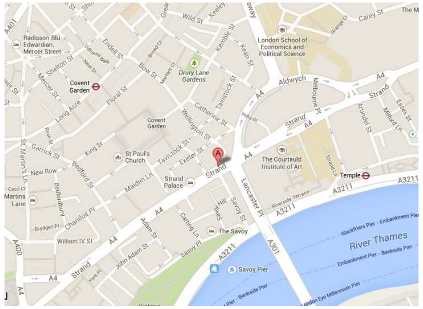
- Web search (Google bing)
  - Search ground are billions of documents on millions of computers
  - issues: spidering; efficient indexing and search; malicious manipulation to boost search engine rankings
  - Link analysis covered in Lecture 8
- Enterprise and institutional search (PubMed LexisNexis)
  - 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

A short history of IR

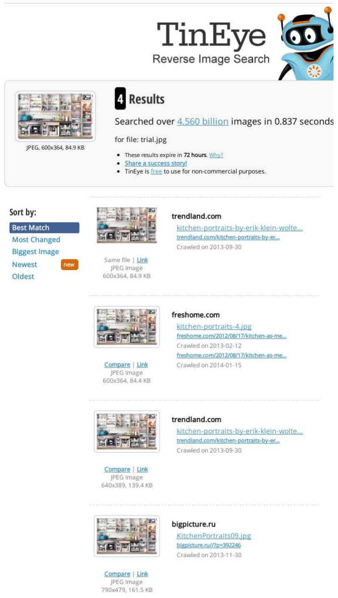


IR for non-textual media





Similarity Searches



JPEG, 600x364, 84.9 KB

4 Results

Searched over 4.560 billion images in 0.837 seconds

for file: trial.jpg

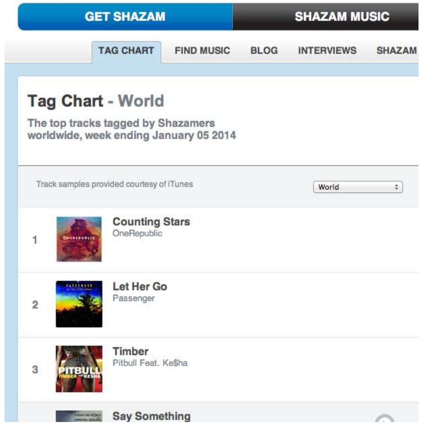
- These results expire in **72 hours**. Why?
- Share a success story!
- TinEye is free to use for non-commercial purposes.

trendland.com
kitchen-portraits-by-erik-klein-wolte...
trendland.com/kitchen-portraits-by-er...
Crawled on 2013-09-30
Same file | Link
JPEG Image
600x364, 84.9 KB

freshome.com
kitchen-portraits-4.jpg
freshome.com/2012/08/17/kitchen-as-me...
Crawled on 2013-02-12
freshome.com/2012/08/17/kitchen-as-me...
Crawled on 2014-01-15
Compare | Link
JPEG Image
600x364, 84.4 KB

trendland.com
kitchen-portraits-by-erik-klein-wolte...
trendland.com/kitchen-portraits-by-er...
Crawled on 2013-09-30
Compare | Link
JPEG Image
640x389, 139.4 KB

bigpicture.ru
KitchenPortraits09.jpg
bigpicture.ru//?p=392246
Crawled on 2013-11-30
Compare | Link
JPEG Image
790x479, 161.5 KB



Tag Chart - World

The top tracks tagged by Shazamers worldwide, week ending January 05 2014

Track samples provided courtesy of iTunes

1 Counting Stars
OneRepublic

2 Let Her Go
Passenger

3 Timber
Pitbull Feat. Ke$ha

Say Something

Areas of IR

- “Ad hoc” retrieval and classification (lectures 1-5)
- web retrieval (lecture 8)
- Support for browsing and filtering document collections:
  - Evaluation lecture 6)
  - Clustering (lecture 7)
- Further processing a set of retrieved documents, e.g., by using natural language processing
  - Information extraction
  - Summarisation
  - Question answering

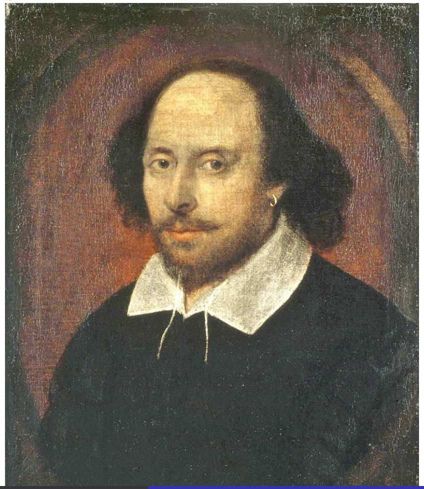
Overview

1 Motivation
- Definition of “Information Retrieval”
- IR: beginnings to now

2 First Boolean Example
- Term-Document Incidence matrix
- The inverted index
- Processing Boolean Queries
- Practicalities of Boolean Search

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 colletions, 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.

Query “Brutus AND Caesar AND NOT Calpunia”

We compute the results for our query as the bitwise AND between vectors for Brutus, Caesar and complement (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 complement (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 complement (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

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.

Bigger collections

- Consider N=10^6 documents, each with about 1000 tokens
- 10^9 tokens at avg 6 Bytes per token ⇒ 6GB
- Assume there are M=500,000 distinct terms in the collection
- Size of incidence matrix is then 500,000 $\times 10^6$
- Half a trillion 0s and 1s

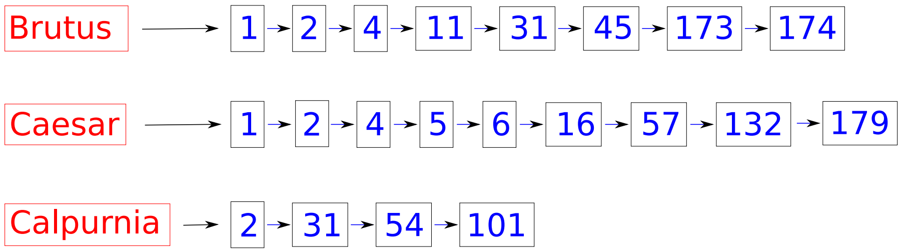
Can’t build the Term-Document incidence matrix

- 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.

The inverted index

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

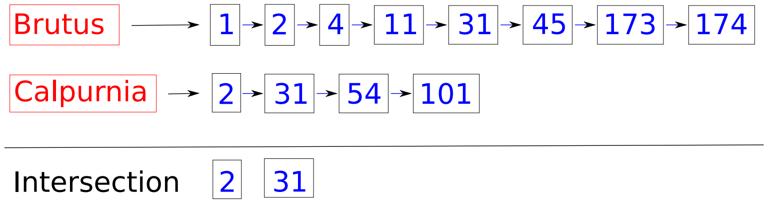
Calpurnia → 2 → 31 → 54 → 101

Processing Boolean Queries: conjunctive queries

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

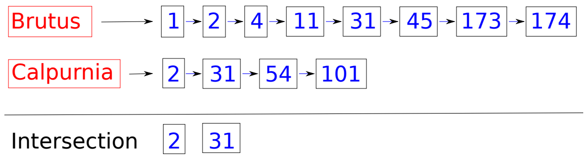
Calpurnia → 2 → 31 → 54 → 101

Intersection 2 31

Note: this only works if postings lists are sorted

Algorithm for intersection of two postings

```
INTERSECT (p1, p2)
  answer $\leftarrow$ <>
  while p1 $\neq$ NIL and p2 $\neq$ NIL
  do if docID(p1) = docID(p2)
       then ADD (answer, docID(p1))
            p1 $\leftarrow$ next(p1)
            p2 $\leftarrow$ next(p2)
       if docID(p1) $<$ docID(p2)
            then p1$\leftarrow$ next(p1)
            else p2$\leftarrow$ next(p2)
   return answer
```



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

Calpurnia → 2 → 31 → 54 → 101

Intersection 2 31

Complexity of the Intersection Algorithm

- 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 $O(N)$

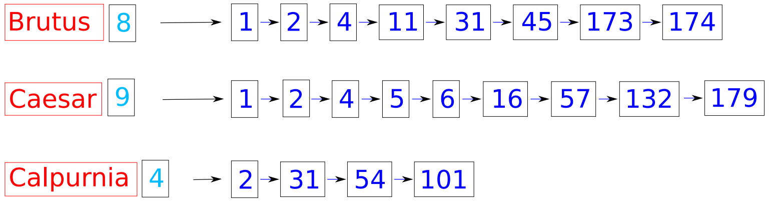
Query Optimisation: conjunctive terms

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

(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

Practical Boolean Search

- Provided by large commercial information providers 1960s-1990s
- Complex query language; complex and long queries
- Extended Boolean retrieval models with additional operators – proximity operators
- Proximity operator: two terms must occur close together in a document (in terms of certain number of words, or within sentence or paragraph)
- Unordered results...

Examples

- Westlaw : Largest commercial legal search service – 500K subscribers
- Medical search
- Patent search
- Useful when expert queries are carefully defined and incrementally developed

Does Google use the Boolean Model?

On Google, the default interpretation of a query [w_1 w_2 ... w_n] is w_1 AND w_2 AND ... AND w_n

- Cases where you get hits which don't contain one of the w$-i$:
  - Page contains variant of w_i (morphology, misspelling, synonym)
  - long query (n is large)
  - Boolean expression generates very few hits
  - w_i was in the *anchor text*
- Google also *ranks* the result set
  - Simple Boolean Retrieval returns matching documents in no particular order.
  - Google (and most well-designed Boolean engines) rank hits according to some estimator of relevance

Reading

- Manning, Raghavan, Schütze: Introduction to Information Retrieval (MRS), chapter 1

Lecture 2: Data structures and Algorithms for Indexing

Information Retrieval
Computer Science Tripos Part II

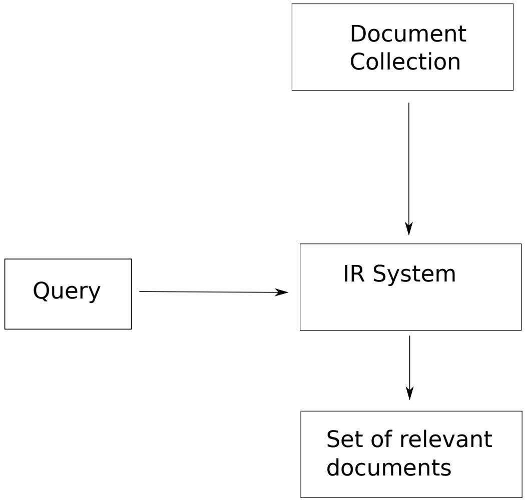
Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

2017

[1] Adapted from Simone Teufel's original slides

IR System Components



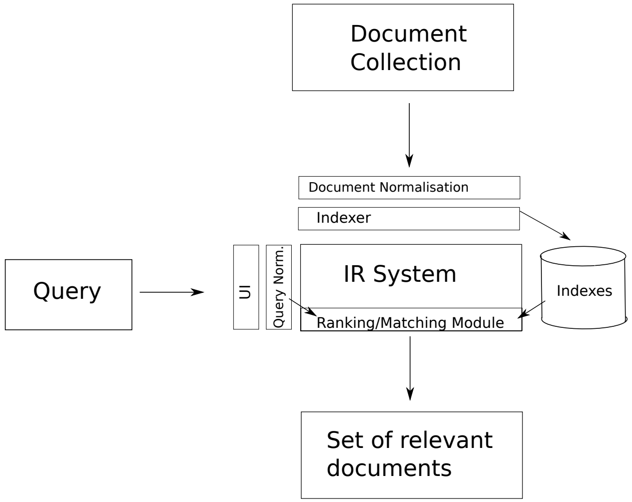
Document Collection

Query

IR System

Set of relevant documents

IR System Components



Document Collection

Document Normalisation

Indexer

Query

UI

Query Norm.

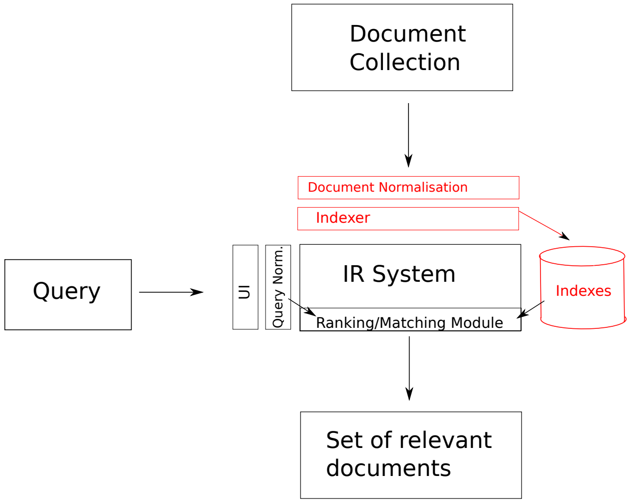
IR System

Ranking/Matching Module

Indexes

Set of relevant documents

IR System Components



Document Collection

Document Normalisation

Indexer

Query

UI

Query Norm.

IR System

Ranking/Matching Module

Indexes

Set of relevant documents

Today: The indexer

Overview

1. Index construction
   - Postings list and Skip lists
   - Single-pass Indexing
2. Document and Term Normalisation
   - Documents
   - Terms
   - Reuter RCV1 and Heap's Law

Index construction

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.

Example: index creation by sorting

Doc 1:
I did enact Julius Caesar: I was killed i' the Capitol;Brutus killed me.

⟹ Tokenisation

Doc 2:
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

⟹ Tokenisation

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

⟹ Sorting

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

Index creation; grouping step ("uniq")

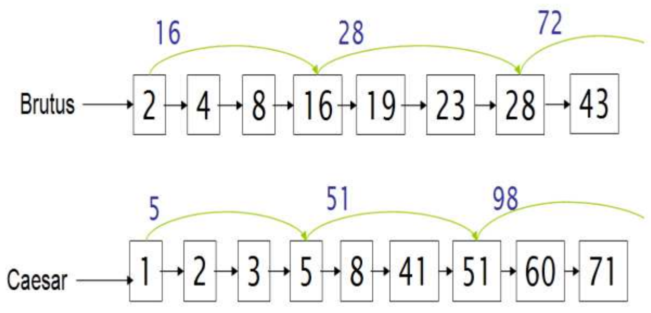
Term	doc. freq.		Postings list
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

- Primary sort by term (dictionary)
- Secondary sort (within postings list) by document ID
- Document frequency (= length of postings list):
  - for more efficient Boolean searching (later today)
  - for term weighting (lecture 4)
- keep Dictionary in memory
- keep Postings List (much larger) on disk

Data structures for Postings Lists

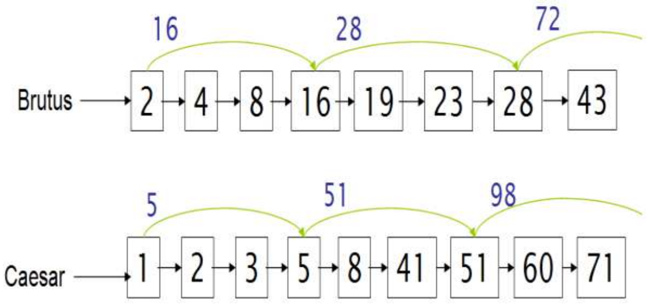
- Singly linked list
  - Allow cheap insertion of documents into postings lists (e.g., when recrawling)
  - Naturally extend to skip lists for faster access
- Variable length array
  - Better in terms of space requirements
  - Also better in terms of time requirements if memory caches are used, as they 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



- 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.

Tradeoff Skip Lists



- 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.

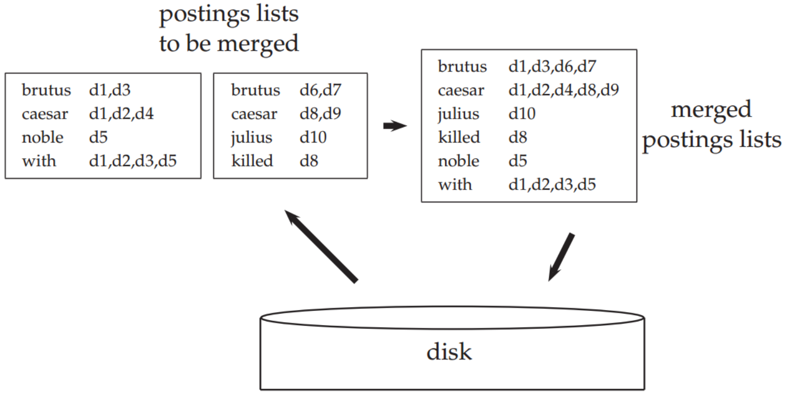
Algorithm: single-pass in-memory indexing or SPIMI

- 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 (1)

- 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 (2)



postings lists to be merged

brutus	d1,d3
caesar	d1,d2,d4
noble	d5
with	d1,d2,d3,d5

brutus	d6,d7
caesar	d8,d9
julius	d10
killed	d8

brutus	d1,d3,d6,d7
caesar	d1,d2,d4,d8,d9
julius	d10
killed	d8
noble	d5
with	d1,d2,d3,d5

merged postings lists

disk

Single-pass in-memory indexing (3)

- 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 and you can read about it in the book (Chapter 4.2)

Overview

1. Index construction
   - Postings list and Skip lists
   - Single-pass Indexing
2. Document and Term Normalisation
   - Documents
   - Terms
   - Reuter RCV1 and Heap's Law

Document and Term Normalisation

- 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

- 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

- 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

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 (`&amp`)

Format/Language: Complications

- A single index usually contains terms of several languages.
- Documents or their components can contain multiple languages/format, for instance a French email with a Spanish pdf attachment
- What is the document unit for indexing?
  - a file?
  - an email?
  - an email with 5 attachments?
  - an email thread?
- Answering the question "What is a document?" is not trivial.
- Smaller units raise precision, drop recall
- Also might have to deal with XML/hierarchies of HTML documents etc.

Normalisation

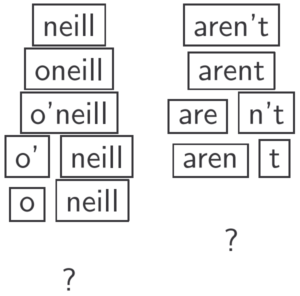
- Need to normalise words in the indexed text as well as query terms to the same form
- Example: We want to match U.S.A. to USA
- We most commonly implicitly define equivalence classes of terms.
- Alternatively, we could do asymmetric expansion:

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

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

Tokenisation

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



neill

oneill

o'neill

o' neill

o neill

?

aren't

arent

are n't

aren t

?

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

- 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

Numbers

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.

Chinese: No Whitespace

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18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

和尚

- As one word, means “monk”
- As two words, means “and” and “still”

Other cases of “no whitespace”: Compounding

Compounding in Dutch, German, Swedish

German

Lebensversicherungsgesellschaftsangestellter
leben+s+versicherung+s+gesellschaft+s+angestellter

Other cases of “no whitespace”: Agglutination

“Agglutinative” languages do this not just for compounds:

Inuit

tusaatsiarunnangittualuujunga
(= “I can’t hear very well”)

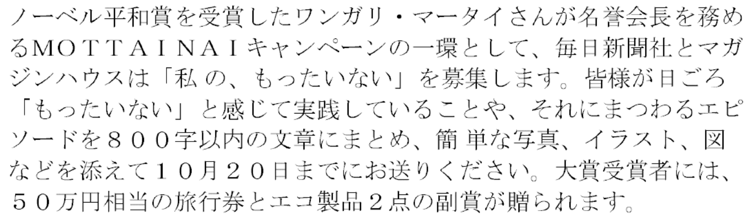
Finnish

epäjärjestelmällistyttämättömyydellänsäkäänköhän
(= “I wonder if – even with his/her quality of not having been made unsystematized”)

Turkish

Çekoslovakyalılaştıramadıklarımızdanmşçasına
(= “as if you were one of those whom we could not make resemble the Czechoslovacian people”)

Japanese

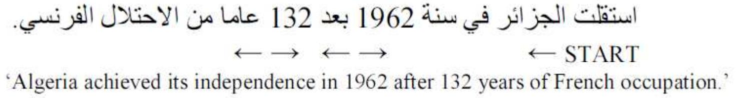


ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私 の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡 単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

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

Arabic script and bidirectionality

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



استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

'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

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

> peña = cliff, pena = sorrow (Spanish)

- Main questions: will users apply it when querying?

Case Folding

- 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 regardness of correct capitalisation.

Stop words

- 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.

More equivalence classing

- Thesauri: semantic equivalence, car = automobile
- Soundex: phonetic equivalence, Muller = Mueller; lecture 3

Lemmatisation

- 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

- 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

- 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).

Stemming: Representation of a word

[C] (VC){m}[V]

C : one or more adjacent consonants
V : one or more adjacent vowels

[ ] : optionality
( ) : group operator
{**x**} : repetition x times
m : the "measure" of a word

shoe	$[\text{sh}]_C[\text{oe}]_V$	m=0
Mississippi	$[\text{M}]_C([\text{i}]_V[\text{ss}]_C)([\text{i}]_V[\text{ss}]_C)([\text{i}]_V[\text{pp}]_C)[\text{i}]_V$	m=3
ears	$([\text{ea}]_V[\text{rs}]_C)$	m=1

Notation: measure *m* is calculated on the word **excluding** the suffix of the rule under consideration

Porter stemmer: selected rules

SSES → SS
IES → I
SS → SS
S →

caresses → caress
cares → care

(m>0) EED →
EE

feed → feed
agreed → agree
BUT: freed, succeed

Porter Stemmer: selected rules

(*v*) ED →

plastered → plaster
bled → bled

Three stemmers: a comparison

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

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries and decreases it for others.

Example queries where stemming helps

tartan sweaters → sweater, sweaters
sightseeing tour san francisco → tour, tours

Example queries where stemming hurts

operational research → “oper” = operates, operatives, operate, operation, operational, operative

operating system → operates, operatives, operate, operation, operational, operative

operative dentistry → operates, operatives, operate, operation, operational, operative

Phrase Queries

- 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

Biword indexes

- 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.

Longer phrase queries

- 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

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

Positional indexes

- 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

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

- 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 search

- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) "employment" in document and (ii) "place" in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
- This is important for dynamic summaries etc.

Proximity intersection

```
PositionalIntersect(p1, p2, k)
answer ←<>
while p1 6= nil and p2 6= nil
do if docID(p1) = docID(p2)
    then l ← <>
       pp1 ← positions(p1)
       pp2 ← positions(p2)
       while pp1 6= nil
       do while pp2 6= nil
          do if |pos(pp1)  pos(pp2)| ≤ k
              then Add(l , pos(pp2))
              else if pos(pp2) > pos(pp1)
                 then break
              pp2 ← next(pp2)
           while l ≠<> and |l [0]  pos(pp1)| > k
           do Delete(l [0])
           for each ps  l
           do Add(answer , hdocID(p1), pos(pp1), psi)
           pp1 ← next(pp1)
        p1 ← next(p1)
         p2 ← next(p2)
     else if docID(p1) < docID(p2)
        then p1 ← next(p1)
        else p2 ← next(p2)
return answer
```

Combination scheme

- 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

- 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

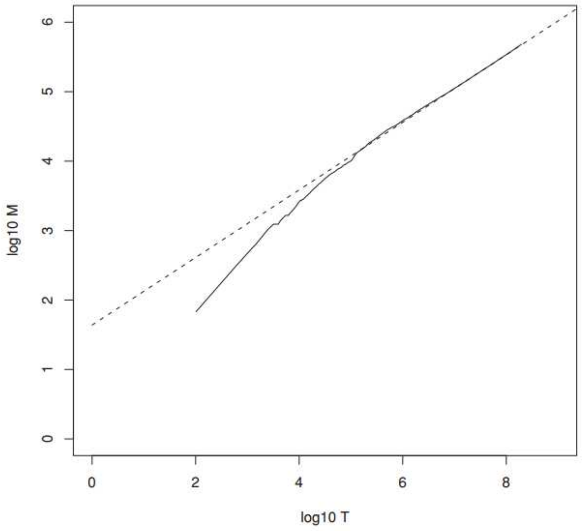
Effect of preprocessing for Reuters

	word types (terms)			non-positional postings			positional postings (word tokens)		
size of	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

How big is the term vocabulary?

- That is, how many distinct words are there?
- Can we assume there is an upper bound?
- Not really: At least $70^{20} \approx 10^{37}$ different words of length 20.
- The vocabulary will keep growing with collection size.
- Heaps' law: $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection.
- Typical values for the parameters k and b are: $30 \leq k \leq 100$ and $b \approx 0.5$.
- Heaps' law is linear in log-log space.
  - It is the simplest possible relationship between collection size and vocabulary size in log-log space.
  - Empirical law

Heaps' law for Reuters



Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1. For these data, the dashed line $\log_{10} M = 0.49 * \log_{10} T + 1.64$ is the best least squares fit. Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$.

Empirical fit for Reuters

- Good, as we just saw in the graph.
- Example: for the first 1,000,020 tokens Heaps' law predicts 38,323 terms:

$$44 \times 1{,}000{,}020^{0.49} \approx 38{,}323$$

- The actual number is 38,365 terms, very close to the prediction.
- Empirical observation: fit is good in general.

Take-away

- Understanding of the basic unit of classical information retrieval systems: words and documents: What is a document, what is a term?
- Tokenization: how to get from raw text to terms (or tokens)
- More complex indexes for phrases

Reading

- MRS Chapter 2.2
- MRS Chapter 2.4
- MRS Chapter 4.3

Lecture 3: Index Representation and Tolerant Retrieval

Information Retrieval
Computer Science Tripos Part II

Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

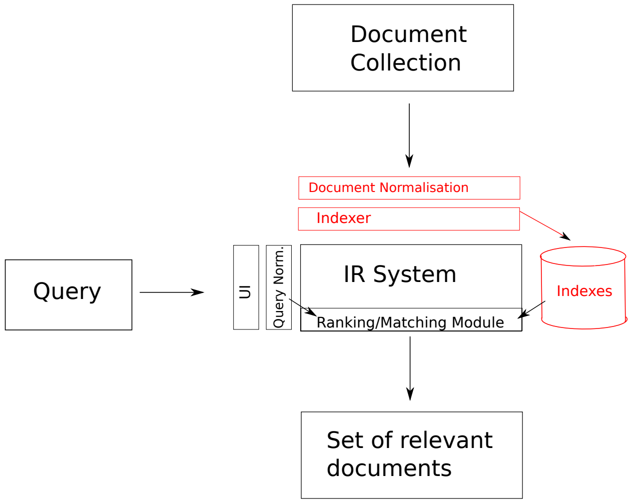
2017

[1] Adapted from Simone Teufel's original slides

Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

IR System components



Document Collection

Document Normalisation

Indexer

Query

UI

Query Norm.

IR System

Ranking/Matching Module

Indexes

Set of relevant documents

Last time: The indexer

Type/token distinction

- Token an instance of a word or term occurring in a document
- Type an equivalence class of tokens

> In June, the dog likes to chase the cat in the barn.

- 12 word tokens
- 9 word types

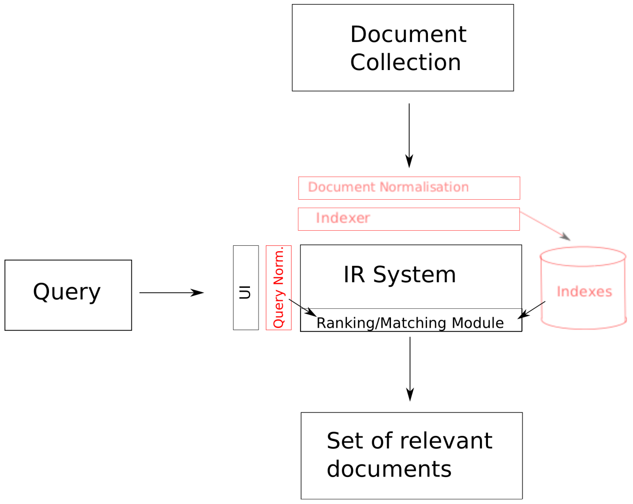
Problems with equivalence classing

- A term is an equivalence class of tokens.
- How do we define equivalence classes?
- Numbers (3/20/91 vs. 20/3/91)
- Case folding
- Stemming, Porter stemmer
- Morphological analysis: inflectional vs. derivational
- Equivalence classing problems in other languages

Positional 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
- Example query: “to$_1$ be$_2$ or$_3$ not$_4$ to$_5$ be$_6$”
- With a positional index, we can answer
  - phrase queries
  - proximity queries

IR System components



Today: more indexing, some query normalisation

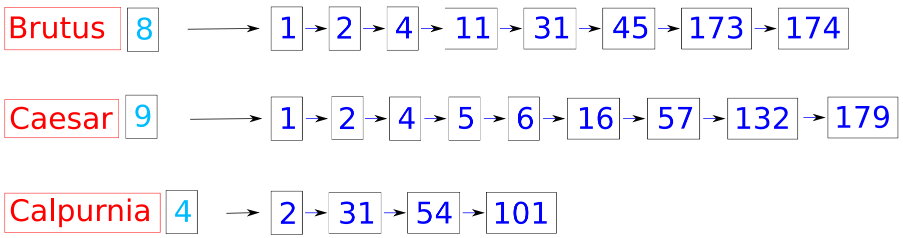
Upcoming

- 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

Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

Inverted Index



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

- The dictionary is the data structure for storing the term vocabulary.
- Term vocabulary: the data
- Dictionary: the data structure for storing the term vocabulary

Dictionaries

- 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?

Data structures for looking up terms

- 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?

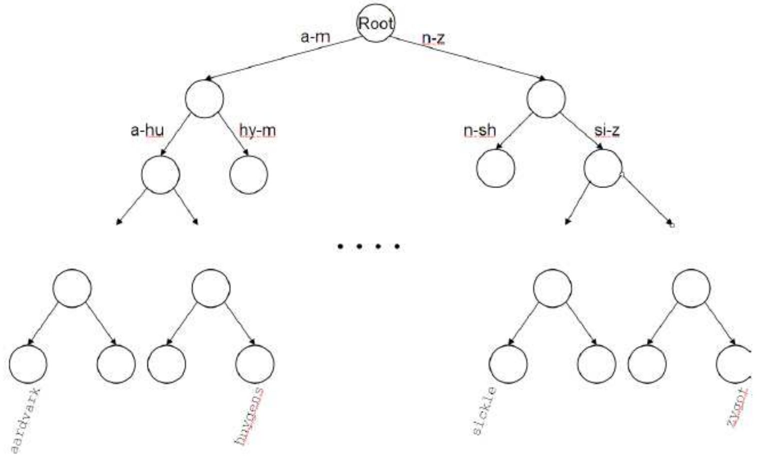
Hashes

- 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

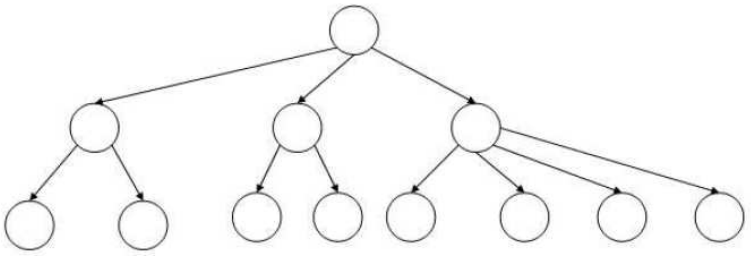
Trees

- 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



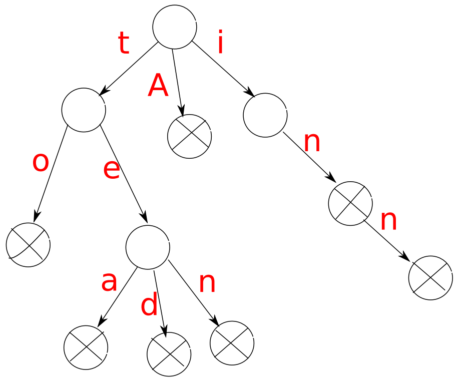
B-tree



Trie

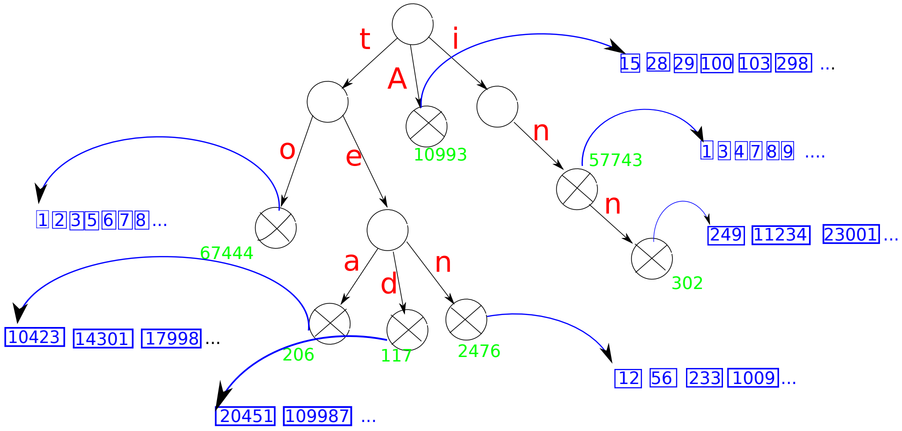
- An ordered tree data structure that is used to store an associative array
- The keys are strings
- The key associated with a node is inferred from the position of a node in the tree
  - Unlike in binary search trees, where keys are stored in nodes.
- Values are associated only with with leaves and 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



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

Trie with postings



Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

Wildcard queries

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

hel*o

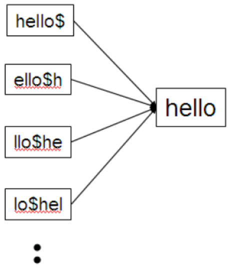
- We could look up “hel*” and “*o” in the tries as before and intersect the two term sets.
  - Expensive
- Alternative: permuterm index
- Basic idea: Rotate every wildcard query, so that the * occurs at the end.
- Store each of these rotations in the dictionary (trie)

Permuterm index

For term hello: add

hello$, ello$h, llo$he, lo$hel, 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).

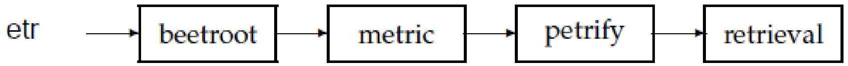
k-gram indexes

- 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

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

- 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.

Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

Spelling correction

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

- 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

- **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

		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

Edit Distance: Four cells

			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**

Each cell of Levenshtein matrix

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

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

		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

Edit distance OSLO–SNOW is 3! How do I read out the editing operations that transform OSLO into SNOW?

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

Using edit distance for spelling correction

- 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.

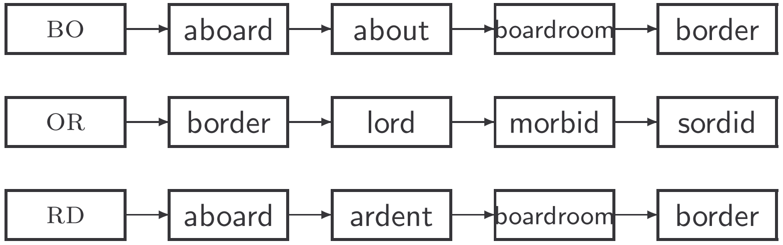
k-gram indexes for spelling correction

- 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



Context-sensitive Spelling correction

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

- 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

- 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

Reading

- Wikipedia article "trie"
- MRS chapter 3.1, 3.2, 3.3

Lecture 4: Term Weighting and the Vector Space Model

Information Retrieval
Computer Science Tripos Part II

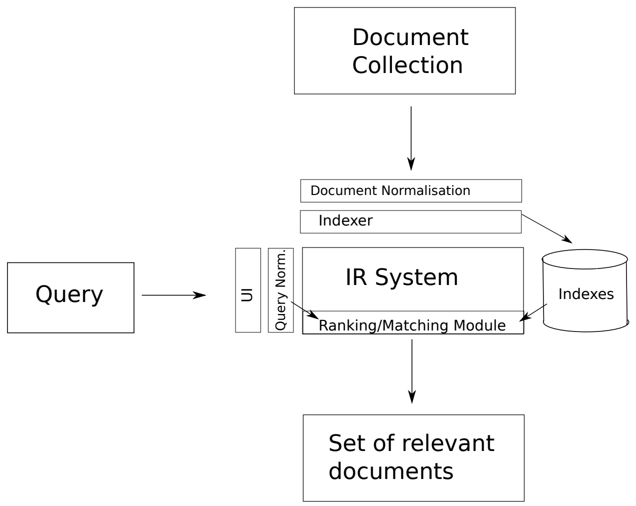
Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

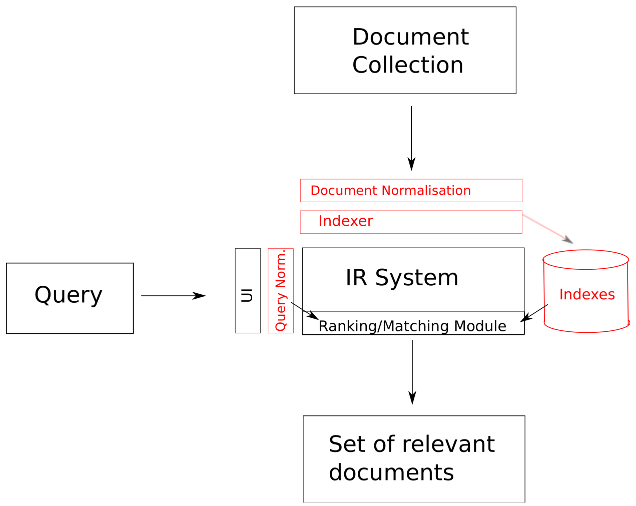
2017

[1] Adapted from Simone Teufel's original slides

IR System Components

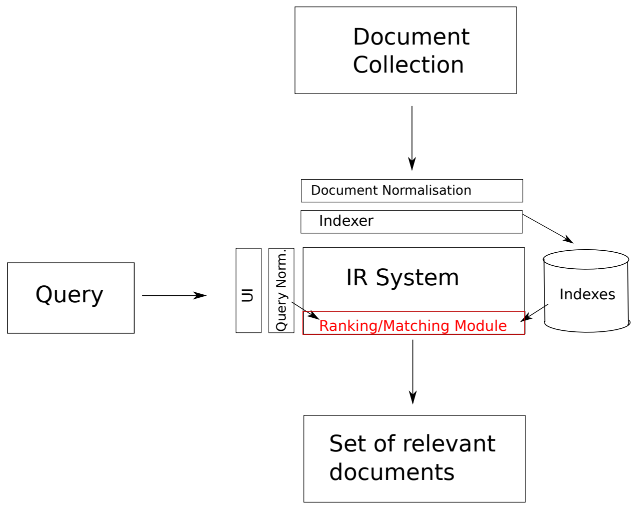


IR System Components



Finished with indexing, query normalisation

IR System Components



Today: the matcher

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Recap: Tolerant Retrieval

- What to do when there is no exact match between query term and document term?
- Dictionary as hash, B-tree, trie
- Wildcards via permuterm
- and k-gram index
- k-gram index and edit-distance for spelling correction

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)

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

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:

$$\textsc{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

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

Jaccard coefficient: Example

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

Query

"ides of March"

Document

"Caesar died in March"

- $\textsc{jaccard}(q, d) = 1/6$

What's wrong with Jaccard?

- 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.

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Binary incidence matrix

	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

	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 $\in \mathbb{N}^{|V|}$.

Bag of words model

- 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.

Term frequency tf

- 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.

Instead of raw frequency: Log frequency weighting

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

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

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

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

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

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

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Frequency in document vs. frequency in collection

- 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

- 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

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 $k = -1$)
- Example of a power law

Zipf's Law: Examples from 5 Languages

Top 10 most frequent words in some large language samples:

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

Zipf's law: Rank × Frequency ∼ Constant

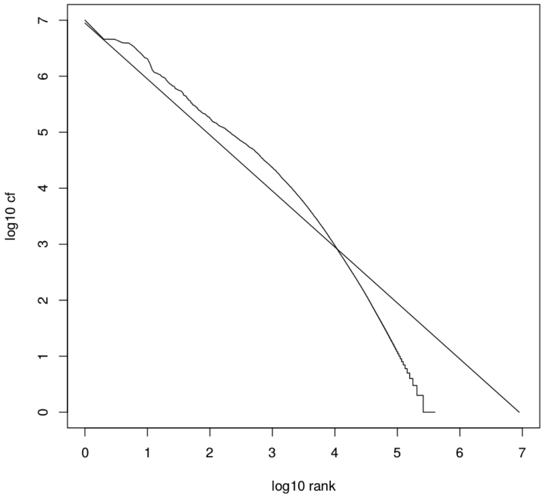
English: Rank R	Word	Frequency f	$R \times f$
10	he	877	8770
20	but	410	8200
30	be	294	8820
800	friends	10	8000
1000	family	8	8000

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

Other collections (allegedly) obeying power laws

- Sizes of settlements
- Frequency of access to web pages
- Income distributions amongst top earning 3% individuals
- Korean family names
- Size of earth quakes
- Word senses per word
- Notes in musical performances
- . . .

Zipf's law for Reuters



Fit is not great.

Desired weight for rare terms

- 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.

Desired weight for frequent terms

- 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

- 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.

idf weight

- 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:

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

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

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

Examples for idf

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

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

Effect of idf on ranking

- 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

	Collection	Document
Term	frequency	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").

tf-idf weighting

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

tf-idf weight

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

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

Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. Zipf's Law and tf-idf weighting
5. The vector space model

Binary incidence matrix

	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

	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

	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

- 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.

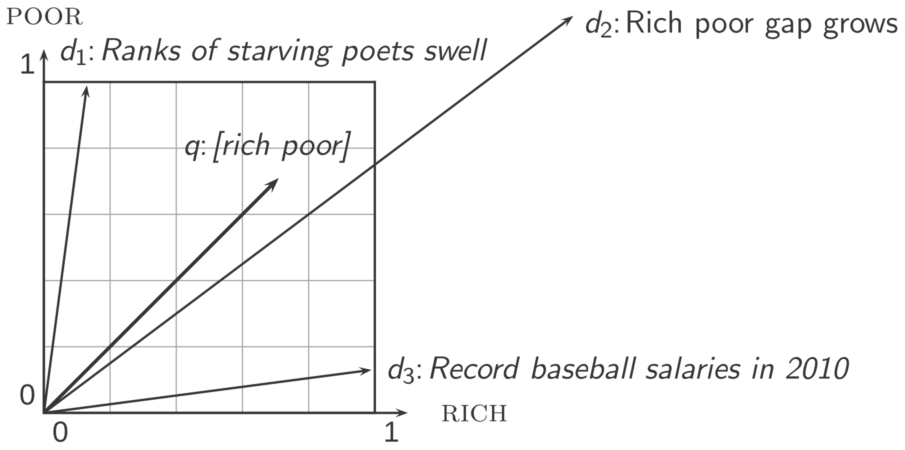
Queries as vectors

- 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?

- 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


POOR
d_1: *Ranks of starving poets swell*
d_2: Rich poor gap grows
q: [rich poor]
d_3: *Record baseball salaries in 2010*
RICH
1
0
0
1

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.

Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document d and append it to itself. Call this document d'. d' is twice as long as d.
- "Semantically" d and d' have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though the Euclidean distance between the two documents can be quite large.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents according to the angle between query and document in decreasing order
  - Rank documents according to cosine(query,document) in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 180^\circ]$

Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – here we use the L_2 norm: $||x||_2 = \sqrt{\sum_i x_i^2}$
- This maps vectors onto the unit sphere . . .
- . . . since after normalization: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have identical vectors after length-normalization.

Cosine similarity between query and document

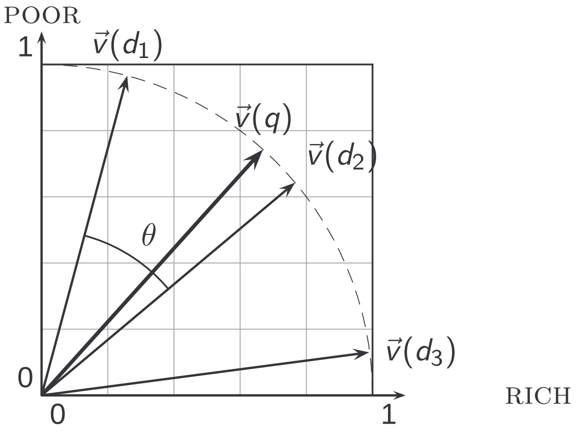
$$\cos(\vec{q},\vec{d}) = \textsc{sim}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- 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 $\vec{d}$.

Cosine for normalized vectors

- 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



Cosine: Example

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

	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

Components of tf-idf weighting

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

Best known combination of weighting options

Default: no weighting

tf-idf example

- 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

tf-idf example: lnc.ltn

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: $\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$

Summary: Ranked retrieval in the vector space model

- 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

Reading

- MRS, Chapter 5.1.2 (Zipf's Law)
- MRS, Chapter 6 (Term Weighting)

Lecture 5: Language Modelling in Information Retrieval and Classification

Information Retrieval
Computer Science Tripos Part II

Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

2017

[1] Adapted from Simone Teufel's original slides

Upcoming today

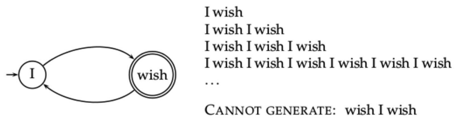
- Query-likelihood method in IR
- Document Language Modelling
- Smoothing
- Classification

Overview

1. Query Likelihood
2. Estimating Document Models
3. Smoothing
4. Naive Bayes Classification

Language Model

- A model for how humans generate language
- Used in many language orientated-tasks (MT, word prediction, IR)
- Usually probabilistic in nature (e.g. multinomial, neural)



I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish
. . .

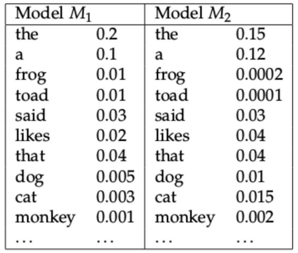
CANNOT GENERATE: wish I wish

► **Figure 12.1** A simple finite automaton and some of the strings in the language it generates. → shows the start state of the automaton and a double circle indicates a (possible) finishing state.

What is a document language model?

- A model for how an author generates a document on a particular topic
- The document itself is just one sample from the model (i.e. ask the author to write the document again and he/she will invariably write something similar, but not exactly the same)
- A probabilistic generative model for documents

Two Document Models



Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
. . .	. . .	. . .	. . .

► **Figure 12.3** Partial specification of two unigram language models.

$$\sum_{t \in V} P(t|M_d) = 1 \tag{1}$$

Query Likelihood Method (I)

- Users often pose queries by thinking of words that are likely to be in *relevant* documents
- The query likelihood approach uses this idea as a principle for ranking documents
- Given a query string q, we rank documents by the likelihood of their document *models* M_d generating q

Query Likelihood Method (II)

$$P(d|q) = P(q|d)P(d)/P(q) \tag{2}$$

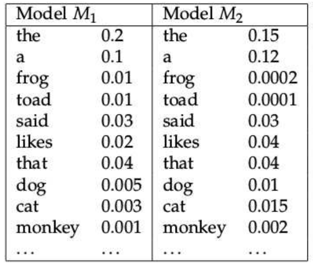
$$P(d|q) \propto P(q|d)P(d) \tag{3}$$

where if we have a uniform prior over $P(d)$ then

$$P(d|q) \propto P(q|d) \tag{4}$$

Note: $P(d)$ is uniform if we have no reason a priori to favour one document over another. Useful priors (based on aspects such as authority, length, novelty, freshness, popularity, click-through rate) could easily be incorporated.

An Example (I)



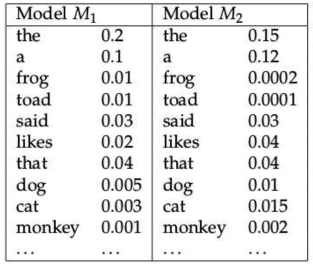
Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
…	…	…	…

▶ **Figure 12.3** Partial specification of two unigram language models.

$$P(frog\ said\ that\ toad\ likes\ frog|M_1) = (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01) \tag{5}$$

$$P(frog\ said\ that\ toad\ likes\ frog|M_2) = (0.0002 \times 0.03 \times 0.04 \times 0.0001 \times 0.04 \times 0.0002) \tag{6}$$

An Example (II)



Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
…	…	…	…

▶ **Figure 12.3** Partial specification of two unigram language models.

$$P(q|M_1) > P(q|M_2) \tag{7}$$

Overview

1. Query Likelihood
2. Estimating Document Models
3. Smoothing
4. Naive Bayes Classification

Documents as samples

- We now know how to rank document models in a theoretically principled manner.
- But how do we estimate the document model for each document?

document 1

click go the shears boys click click click

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125,

Zero probability problem (over-fitting)

- When using maximum likelihood estimates, documents that do not contain *all* query terms will receive a score of zero

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125

Sample query

$P(shears\ boys\ hair|M_d) = 0.0$

What if the query is long?

Make sure no non-zero probabilities

- Only assign a zero probability when something *cannot* happen
- Remember that the document model is a generative explanation
- If a person was to rewrite the document he/she may include *hair* or indeed some other words

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125

Some type of smoothing

click=0.4, go=0.1, the=0.1, shears=0.1, boys=0.1, hair=0.01, man=0.01, the=0.001, bacon=0.0001, ...

Overview

1. Query Likelihood
2. Estimating Document Models
3. Smoothing
4. Naive Bayes Classification

How to smooth

ML estimates

$$\hat{P}(t|M_d) = \frac{tf_t}{|d|} \tag{8}$$

Maximum likelihood estimates

click=0.5, go=0.125, the=0.125, shears=0.125, boys=0.125

Linear Smoothing

$$\hat{P}(t|M_d) = \lambda\frac{tf_t}{|d|} + (1-\lambda)\hat{P}(t|M_c) \tag{9}$$

where λ is a smoothing parameter between 0 and 1, and $\hat{P}(t|M_c) = \frac{cf_t}{|c|}$ is the estimated probability of seeing t in general (i.e. ct_t is the frequency of t in the entire document collection of $|c|$ tokens).

How to smooth

Linear Smoothing

$$\hat{P}(t|M_d) = \lambda\frac{tf_t}{|d|} + (1-\lambda)\frac{cf_t}{|c|} \tag{10}$$

Dirichlet Smoothing has been found to be more effective in IR where λ is $\frac{|d|}{\alpha+|d|}$. Plugging this in yields:

$$\hat{P}(t|M_d) = \frac{|d|}{\alpha+|d|}\frac{tf_t}{|d|} + \frac{\alpha}{\alpha+|d|}\frac{cf_t}{|c|} \tag{11}$$

where α is interpreted as the background mass (pseudo-counts).

Bayesian Intuition

We should have more trust (belief) in ML estimates that are derived from longer documents (see the $\frac{|d|}{\alpha+|d|}$ factor).

Putting this all together

Rank documents according to:

$$P(q|d) = \prod_{t\in q}\left(\frac{|d|}{\alpha+|d|}\frac{tf_t}{|d|} + \frac{\alpha}{\alpha+|d|}\frac{cf_t}{|c|}\right) \tag{12}$$

or

$$log\ P(q|d) = \sum_{t\in q} log\left(\frac{|d|}{\alpha+|d|}\frac{tf_t}{|d|} + \frac{\alpha}{\alpha+|d|}\frac{cf_t}{|c|}\right) \tag{13}$$

Pros and Cons

- It is principled, intuitive, simple, and extendable
- Aspects of *tf* and *idf* are incorporated quite naturally
- It is computationally efficient for large scale corpora
- More complex language models (markov-models) can be adopted and priors can be added
- But more complex models usually involve storing more parameters (and doing more computation)

- Both documents and queries are modelled as simple strings of symbols
- No formal treatment of relevance
- Therefore model does not handle relevance feedback automatically

Extensions

- Relevance-based language models (very much related to Naive-Bayes classification) incorporate the idea of relevance and are useful for capturing feedback
- Treating the query as being drawn from a query model (useful for long queries)
- Markov-chain models for document modelling
- Use different generative distributions (e.g. replacing the multinomial with neural models)

Overview

1. Query Likelihood
2. Estimating Document Models
3. Smoothing
4. Naive Bayes Classification

The Naive Bayes classifier

- The Naive Bayes classifier is a probabilistic classifier.
- We compute the probability of a document d being in a class c as follows:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

- n_d is the length of the document. (number of tokens)
- $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c
- $P(t_k|c)$ as a measure of how much evidence t_k contributes that c is the correct class.
- $P(c)$ is the prior probability of c.
- If a document's terms do not provide clear evidence for one class vs. another, we choose the c with highest $P(c)$.

Maximum a posteriori class

- Our goal in Naive Bayes classification is to find the "best" class.
- The best class is the most likely or maximum a posteriori (MAP) class c_{map}:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \ \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

Taking the log

- Multiplying lots of small probabilities can result in floating point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, we can sum log probabilities instead of multiplying probabilities.
- Since log is a monotonic function, the class with the highest score does not change.
- So what we usually compute in practice is:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]$$

Naive Bayes classifier

- Classification rule:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]$$

- Simple interpretation:
  - Each conditional parameter $\log \hat{P}(t_k|c)$ is a weight that indicates how good an indicator t_k is for c.
  - The prior $\log \hat{P}(c)$ is a weight that indicates the relative frequency of c.
  - The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class.
  - We select the class with the most evidence.

Parameter estimation take 1: Maximum likelihood

- Estimate parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$ from train data: How?
- Prior:

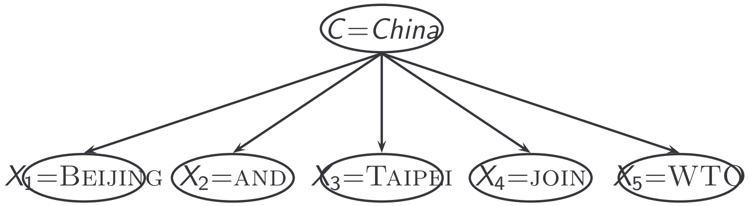
$$\hat{P}(c) = \frac{N_c}{N}$$

- N_c: number of docs in class c; N: total number of docs
- Conditional probabilities:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- T_{ct} is the number of tokens of t in training documents from class c (includes multiple occurrences)
- We've made a Naive Bayes independence assumption here: $\hat{P}(t_{k_1}|c) = \hat{P}(t_{k_2}|c)$, independent of positions k_1, k_2

The problem with maximum likelihood estimates: Zeros



C=*China*

X_1=BEIJING X_2=AND X_3=TAIPEI X_4=JOIN X_5=WTO

$$\begin{aligned} P(China|d) \;\propto\;& P(China)\cdot P(\textsc{Beijing}|China)\cdot P(\textsc{and}|China)\\ &\cdot P(\textsc{Taipei}|China)\cdot P(\textsc{join}|China)\cdot P(\mathrm{WTO}|China) \end{aligned}$$

- If WTO never occurs in class China in the train set:

$$\hat{P}(\mathrm{WTO}|China) = \frac{T_{China,\mathrm{WTO}}}{\sum_{t' \in V} T_{China,t'}} = \frac{0}{\sum_{t' \in V} T_{China,t'}} = 0$$

The problem with maximum likelihood estimates: Zeros

- If there are no occurrences of WTO in documents in class China, we get a zero estimate:

$$\hat{P}(\mathrm{WTO}|China) = \frac{T_{China,\mathrm{WTO}}}{\sum_{t' \in V} T_{China,t'}} = 0$$

- $\rightarrow$ We will get $P(China|d) = 0$ for any document that contains WTO!

To avoid zeros: Add-one smoothing

- Before:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- Now: Add one to each count to avoid zeros:

$$\hat{P}(t|c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'}) + B}$$

- B is the number of bins – in this case the number of different words or the size of the vocabulary $|V| = M$

Example

	docID	words in document	in c = *China*?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

- Estimate parameters of Naive Bayes classifier
- Classify test document

$|text_c| = 8$
$|text_{\overline{c}}| = 3$
B=6 (vocabulary)

Example: Parameter estimates

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$

Conditional probabilities:

$$\begin{aligned}
\hat{P}(\textsc{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\textsc{Tokyo}|c) = \hat{P}(\textsc{Japan}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\textsc{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\textsc{Tokyo}|\overline{c}) = \hat{P}(\textsc{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.

Example: Classification

$$\begin{aligned}
\hat{P}(c|d_5) &\propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003 \\
\hat{P}(\overline{c}|d_5) &\propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001
\end{aligned}$$

Thus, the classifier assigns the test document to $c = China$. The reason for this classification decision is that the three occurrences of the positive indicator CHINESE in d_5 outweigh the occurrences of the two negative indicators JAPAN and TOKYO.

Time complexity of Naive Bayes

mode	time complexity
training	$\Theta(\lvert\mathbb{D}\rvert L_{\text{ave}} + \lvert\mathbb{C}\rvert\lvert V\rvert)$
testing	$\Theta(L_{\text{a}} + \lvert\mathbb{C}\rvert M_{\text{a}}) = \Theta(\lvert\mathbb{C}\rvert M_{\text{a}})$

- L_{ave}: average length of a training doc, L_{a}: length of the test doc, M_{a}: number of distinct terms in the test doc, $\mathbb{D}$: training set, V: vocabulary, $\mathbb{C}$: set of classes
- $\Theta(|\mathbb{D}|L_{\text{ave}})$ is the time it takes to compute all counts. Note that $|\mathbb{D}|L_{\text{ave}}$ is T, the size of our collection.
- $\Theta(|\mathbb{C}||V|)$ is the time it takes to compute the conditional probabilities from the counts.
- Generally: $|\mathbb{C}||V| < |\mathbb{D}|L_{\text{ave}}$
- Test time is also linear (in the length of the test document).
- Thus: Naive Bayes is linear in the size of the training set (training) and the test document (testing). This is optimal.

Naive Bayes is not so naive

- Multinomial model violates two independence assumptions and yet...
- Naive Bayes has won some competitions (e.g., KDD-CUP 97; prediction of most likely donors for a charity)
- More robust to nonrelevant features than some more complex learning methods
- More robust to concept drift (changing of definition of class over time) than some more complex learning methods
- Better than methods like decision trees when we have many equally important features
- A good dependable baseline for text classification (but not the best)
- Optimal if independence assumptions hold (never true for text, but true for some domains)
- Very fast; low storage requirements

Not covered

- Derivation of NB formula
- Evaluation of text classification

Summary

- Query-likelihood as a general principle for ranking documents in an unsupervised manner
  - Treat queries as strings
  - Rank documents according to their models
- Document language models
  - Know the difference between the document and the document model
  - Multinomial distribution is simple but effective
- Smoothing
  - Reasons for, and importance of, smoothing
  - Dirichlet (Bayesian) smoothing is very effective
- Classification
  - Text classification is supervised learning
  - Naive Bayes: simple baseline text classifier

Reading

- Manning, Raghavan, Schütze: Introduction to Information Retrieval (MRS), chapter 12: Language models for information retrieval
- MRS chapters 13.1-13.4 for text classification

Lecture 6: Evaluation

Information Retrieval
Computer Science Tripos Part II

Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

2017

[1] Adapted from Simone Teufel's original slides

Overview

1. Recap/Catchup
2. Introduction
3. Unranked evaluation
4. Ranked evaluation
5. Benchmarks
6. Other types of evaluation

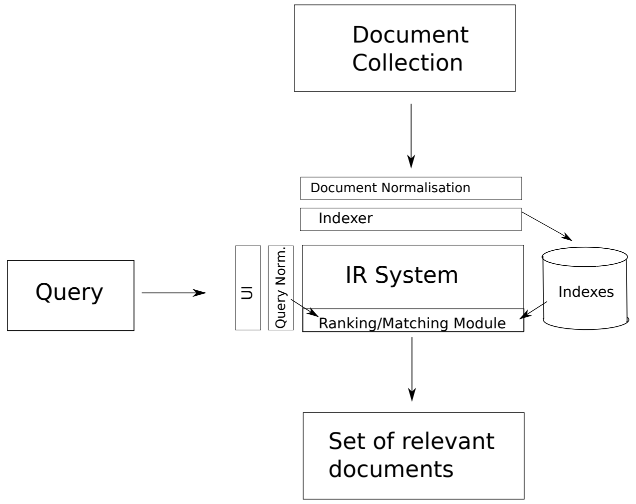
Overview

1. Recap/Catchup
2. Introduction
3. Unranked evaluation
4. Ranked evaluation
5. Benchmarks
6. Other types of evaluation

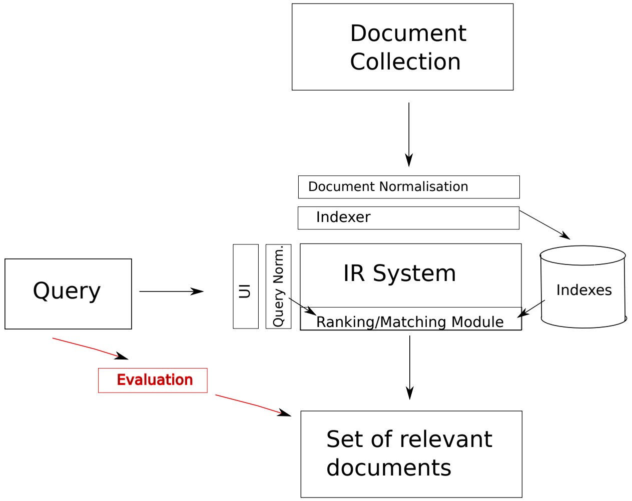
Summary: Ranked retrieval

- In VSM one represents documents and queries as weighted tf-idf vectors
- Compute the cosine similarity between the vectors to rank
- Language models rank based on the probability of a document model generating the query

Today



Today



Document Collection

Document Normalisation

Indexer

Query

UI

Query Norm.

IR System

Ranking/Matching Module

Indexes

Evaluation

Set of relevant documents

Today: how good are the returned documents?

Overview

1. Recap/Catchup
2. Introduction
3. Unranked evaluation
4. Ranked evaluation
5. Benchmarks
6. Other types of evaluation

Measures for a search engine

- How fast does it index?
  - e.g., number of bytes per hour
- How fast does it search?
  - e.g., latency as a function of queries per second
- What is the cost per query?
  - in dollars

Measures for a search engine

- All of the preceding criteria are measurable: we can quantify speed / size / money
- However, the key measure for a search engine is user happiness.
- What is user happiness?
- Factors include:
  - Speed of response
  - Size of index
  - Uncluttered UI
  - We can measure
    - Rate of return to this search engine
    - Whether something was bought
    - Whether ads were clicked
  - Most important: relevance
  - (actually, maybe even more important: it's free)
- Note that none of these is sufficient: blindingly fast, but useless answers won't make a user happy.

Most common definition of user happiness: Relevance

- User happiness is equated with the relevance of search results to the query.
- But how do you measure relevance?
- Standard methodology in information retrieval consists of three elements.
  - A benchmark document collection
  - A benchmark suite of queries
  - An assessment of the relevance of each query-document pair

Relevance: query vs. information need

- Relevance to what? The query?

Information need i

"I am looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine."

- translated into:

Query q

[red wine white wine heart attack]

- So what about the following document:

Document d'

At the heart of his speech was an attack on the wine industry lobby for downplaying the role of red and white wine in drunk driving.

- d' is an excellent match for query q . . .
- d' is not relevant to the information need i.

Relevance: query vs. information need

- User happiness can only be measured by relevance to an information need, not by relevance to queries.
- Sloppy terminology here and elsewhere in the literature: we talk about query–document relevance judgments even though we mean information-need–document relevance judgments.

Overview

1. Recap/Catchup
2. Introduction
3. Unranked evaluation
4. Ranked evaluation
5. Benchmarks
6. Other types of evaluation

Precision and recall

- Precision (P) is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

- Recall (R) is the fraction of relevant documents that are retrieved

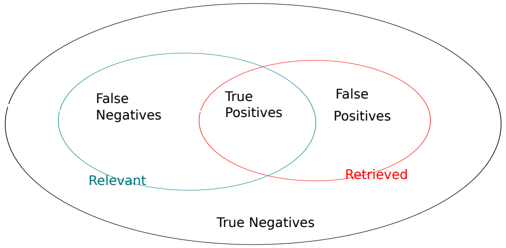
$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

Precision and recall

THE TRUTH

WHAT THE SYSTEM THINKS

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)



$$\begin{aligned} P &= TP/(TP+FP) \\ R &= TP/(TP+FN) \end{aligned}$$

Precision/recall tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.

A combined measure: F

- F allows us to trade off precision against recall.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

- $\alpha \in [0,1]$ and thus $\beta^2 \in [0,\infty]$
- Most frequently used: balanced F with $\beta = 1$ or $\alpha = 0.5$
  - This is the harmonic mean of P and R: $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$

Example for precision, recall, F1

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,060
	80	1,000,040	1,000,120

- $P = 20/(20+40) = 1/3$
- $R = 20/(20+60) = 1/4$
- $F_1 = 2\frac{1}{\frac{1}{\frac{1}{3}}+\frac{1}{\frac{1}{4}}} = 2/7$

Accuracy

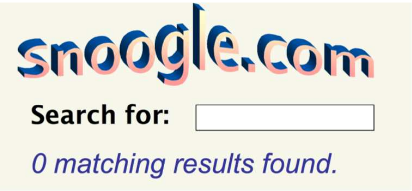
- Why do we use complex measures like precision, recall, and F?
- Why not something simple like accuracy?
- Accuracy is the fraction of decisions (relevant/nonrelevant) that are correct.
- In terms of the contingency table above, accuracy $= (TP + TN)/(TP + FP + FN + TN)$.

Thought experiment

- Compute precision, recall and F_1 for this result set:

	relevant	not relevant
retrieved	18	2
not retrieved	82	1,000,000,000

- The snoogle search engine below always returns 0 results ("0 matching results found"), regardless of the query.



Search for:

0 matching results found.

- Snoogle demonstrates that accuracy is not a useful measure in IR.

Why accuracy is a useless measure in IR

- Simple trick to maximize accuracy in IR: always say no and return nothing
- You then get 99.99% accuracy on most queries.
- Searchers on the web (and in IR in general) want to find something and have a certain tolerance for junk.
- It's better to return some bad hits as long as you return something.
- → We use precision, recall, and F for evaluation, not accuracy.

Recall-criticality and precision-criticality

- Inverse relationship between precision and recall forces general systems to go for compromise between them
- But some tasks particularly need good precision whereas others need good recall:

	Precision-critical task	Recall-critical task
Time	matters	matters less
Tolerance to cases of overlooked information	a lot	none
Information Redundancy	There may be many equally good answers	Information is typically found in only one document
Examples	web search	legal search, patent search

Difficulties in using precision, recall and F

- We should always average over a large set of queries.
    - There is no such thing as a “typical” or “representative” query.
- We need relevance judgments for information-need-document pairs – but they are expensive to produce.
- For alternatives to using precision/recall and having to produce relevance judgments – see end of this lecture.

Overview

1. Recap/Catchup
2. Introduction
3. Unranked evaluation
4. Ranked evaluation
5. Benchmarks
6. Other types of evaluation

Moving from unranked to ranked evaluation

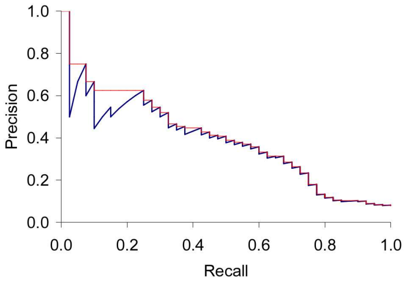
- Precision/recall/F are measures for unranked sets.
- We can easily turn set measures into measures of ranked lists.
- Just compute the set measure for each “prefix”: the top 1, top 2, top 3, top 4 etc results
- This is called Precision/Recall at Rank
- Rank statistics give some indication of how quickly user will find relevant documents from ranked list

Precision/Recall @ Rank

Rank	Doc
1	d_{12}
2	d_{123}
3	d_{4}
4	d_{57}
5	d_{157}
6	d_{222}
7	d_{24}
8	d_{26}
9	d_{77}
10	d_{90}

- Blue documents are relevant
- P@n: P@3=0.33, P@5=0.2, P@8=0.25
- R@n: R@3=0.33, R@5=0.33, R@8=0.66

A precision-recall curve



- Each point corresponds to a result for the top k ranked hits ($k = 1, 2, 3, 4, \ldots$)
- Interpolation (in red): Take maximum of all future points
- Rationale for interpolation: The user is willing to look at more stuff if both precision and recall get better.

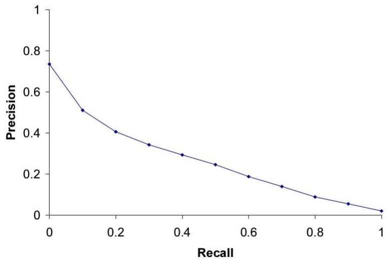
Another idea: Precision at Recall *r*

Rank	S1	S2
1	X	
2		X
3	X	
4		
5		X
6	X	X
7		X
8		X
9	X	
10	X	

→

	S1	S2
p @ r 0.2	1.0	0.5
p @ r 0.4	0.67	0.4
p @ r 0.6	0.5	0.5
p @ r 0.8	0.44	0.57
p @ r 1.0	0.5	0.63

Averaged 11-point precision/recall graph



- Compute interpolated precision at recall levels 0.0, 0.1, 0.2, . . .
- Do this for each of the queries in the evaluation benchmark
- Average over queries
- The curve is typical of performance levels at TREC (more later).

Averaged 11-point precision more formally

$$P_{11\_pt} = \frac{1}{11}\sum_{j=0}^{10}\frac{1}{N}\sum_{i=1}^{N}\tilde{P}_i(r_j)$$

with $\tilde{P}_i(r_j)$ the precision at the jth recall point in the ith query (out of N)

- Define 11 standard recall points $r_j = \frac{j}{10}$: $r_0 = 0,\ r_1 = 0.1\ \ldots\ r_{10} = 1$
- To get $\tilde{P}_i(r_j)$, we can use $P_i(R = r_j)$ directly if a new relevant document is retrieved exacty at r_j
- Interpolation for cases where there is no exact measurement at r_j:

$$\tilde{P}_i(r_j) = \begin{cases} max(r_j \leq r < r_{j+1})P_i(R = r) & \text{if } P_i(R = r) \text{ exists} \\ \tilde{P}_i(r_{j+1}) & \text{otherwise} \end{cases}$$

- Note that $P_i(R = 1)$ can always be measured.
- Worked avg-11-pt prec example for supervisions at end of slides.

Mean Average Precision (MAP)

- Also called "average precision at seen relevant documents"
- Determine precision at each point when a new relevant document gets retrieved
- Use P=0 for each relevant document that was not retrieved
- Determine average for each query, then average over queries

$$MAP = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{Q_j}\sum_{i=1}^{Q_j}P(doc_i)$$

with:

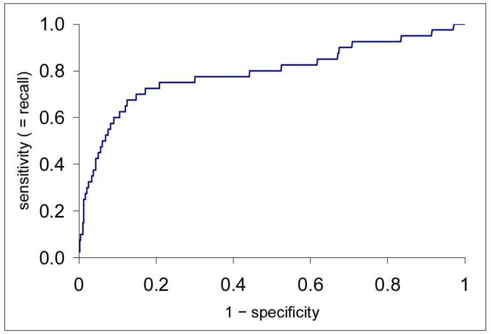
Q_j	number of relevant documents for query j
N	number of queries
$P(doc_i)$	precision at ith relevant document

Mean Average Precision: example ($MAP = \frac{0.564+0.623}{2} = 0.594$)

Query 1		
Rank		$P(doc_i)$
1	X	1.00
2		
3	X	0.67
4		
5		
6	X	0.50
7		
8		
9		
10	X	0.40
11		
12		
13		
14		
15		
16		
17		
18		
19		
20	X	0.25
AVG:		0.564

Query 2		
Rank		$P(doc_i)$
1	X	1.00
2		
3	X	0.67
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15	X	0.2
AVG:		0.623

ROC curve (Receiver Operating Characteristic)



- x-axis: FPR (false positive rate): FP/total actual negatives;
- y-axis: TPR (true positive rate): TP/total actual positives, (also called sensitivity) ≡ recall
- FPR = fall-out = 1 - specificity (TNR; true negative rate)
- But we are only interested in the small area in the lower left corner (blown up by prec-recall graph)

Variance of measures like precision/recall

- For a test collection, it is usual that a system does badly on some information needs (e.g., $P = 0.2$ at $R = 0.1$) and really well on others (e.g., $P = 0.95$ at $R = 0.1$).
- Indeed, it is usually the case that the variance of the same system across queries is much greater than the variance of different systems on the same query.
- That is, there are easy information needs and hard ones.

Overview

1. Recap/Catchup
2. Introduction
3. Unranked evaluation
4. Ranked evaluation
5. Benchmarks
6. Other types of evaluation

What we need for a benchmark

- A collection of documents
  - Documents must be representative of the documents we expect to see in reality.
- A collection of information needs
  - . . . which we will often incorrectly refer to as queries
  - Information needs must be representative of the information needs we expect to see in reality.
- Human relevance assessments
  - We need to hire/pay “judges” or assessors to do this.
  - Expensive, time-consuming
  - Judges must be representative of the users we expect to see in reality.

First standard relevance benchmark: Cranfield

- Pioneering: first testbed allowing precise quantitative measures of information retrieval effectiveness
- Late 1950s, UK
- 1398 abstracts of aerodynamics journal articles, a set of 225 queries, exhaustive relevance judgments of all query-document-pairs
- Too small, too untypical for serious IR evaluation today

Second-generation relevance benchmark: TREC

- TREC = Text Retrieval Conference (TREC)
- Organized by the U.S. National Institute of Standards and Technology (NIST)
- TREC is actually a set of several different relevance benchmarks.
- Best known: TREC Ad Hoc, used for first 8 TREC evaluations between 1992 and 1999
- 1.89 million documents, mainly newswire articles, 450 information needs
- No exhaustive relevance judgments – too expensive
- Rather, NIST assessors' relevance judgments are available only for the documents that were among the top k returned for some system which was entered in the TREC evaluation for which the information need was developed.

Sample TREC Query

<num> Number: 508
<title> hair loss is a symptom of what diseases
<desc> Description:
Find diseases for which hair loss is a symptom.
<narr> Narrative:
A document is relevant if it positively connects the loss of head hair in humans with a specific disease. In this context, "thinning hair" and "hair loss" are synonymous. Loss of body and/or facial hair is irrelevant, as is hair loss caused by drug therapy.

TREC Relevance Judgements

Humans decide which document–query pairs are relevant.

Example of more recent benchmark: ClueWeb09

- 1 billion web pages
- 25 terabytes (compressed: 5 terabyte)
- Collected January/February 2009
- 10 languages
- Unique URLs: 4,780,950,903 (325 GB uncompressed, 105 GB compressed)
- Total Outlinks: 7,944,351,835 (71 GB uncompressed, 24 GB compressed)

Interjudge agreement at TREC

information need	number of docs judged	disagreements
51	211	6
62	400	157
67	400	68
95	400	110
127	400	106

Impact of interjudge disagreement

- Judges disagree a lot. Does that mean that the results of information retrieval experiments are meaningless?
- No.
- Large impact on absolute performance numbers
- Virtually no impact on ranking of systems
- Supposes we want to know if algorithm A is better than algorithm B
- An information retrieval experiment will give us a reliable answer to this question . . .
- . . . even if there is a lot of disagreement between judges.

Overview

1. Recap/Catchup
2. Introduction
3. Unranked evaluation
4. Ranked evaluation
5. Benchmarks
6. Other types of evaluation

Evaluation at large search engines

- Recall is difficult to measure on the web
- Search engines often use precision at top k, e.g., $k = 10$ . . .
- . . . or use measures that reward you more for getting rank 1 right than for getting rank 10 right.
- Search engines also use non-relevance-based measures.
  - Example 1: clickthrough on first result
  - Not very reliable if you look at a single clickthrough (you may realize after clicking that the summary was misleading and the document is nonrelevant) . . .
  - . . . but pretty reliable in the aggregate.
  - Example 2: A/B testing

A/B testing

- Purpose: Test a single innovation
- Prerequisite: You have a large search engine up and running.
- Have most users use old system
- Divert a small proportion of traffic (e.g., 1%) to the new system that includes the innovation
- Evaluate with an “automatic” measure like clickthrough on first result
- Now we can directly see if the innovation does improve user happiness.
- Probably the evaluation methodology that large search engines trust most

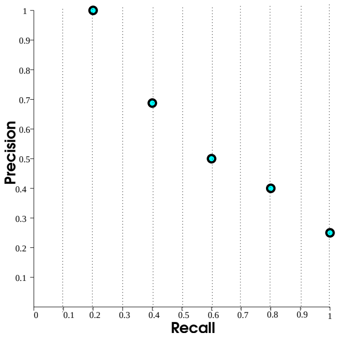
Take-away today

- Focused on evaluation for ad-hoc retrieval
  - Precision, Recall, F-measure
  - More complex measures for ranked retrieval
  - other issues arise when evaluating different tracks, e.g. QA, although typically still use P/R-based measures
- Evaluation for interactive tasks is more involved
- Significance testing is an issue
  - could a good result have occurred by chance?
  - is the result robust across different document sets?
  - slowly becoming more common
  - underlying population distributions unknown, so apply non-parametric tests such as the sign test

Reading

- MRS, Chapter 8

Worked Example avg-11-pt prec: Query 1, measured data points



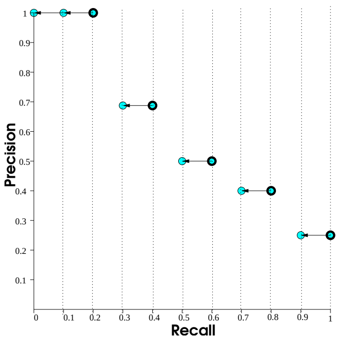
- Blue for Query 1
- Bold Circles measured

Query 1

Rank		R	P	
1	X	0.2	1.00	$\tilde{P}_1(r_2) = 1.00$
2				
3	X	0.4	0.67	$\tilde{P}_1(r_4) = 0.67$
4				
5				
6	X	0.6	0.50	$\tilde{P}_1(r_6) = 0.50$
7				
8				
9				
10	X	0.8	0.40	$\tilde{P}_1(r_8)= 0.40$
11				
12				
13				
14				
15				
16				
17				
18				
19				
20	X	1.0	0.25	$\tilde{P}_1(r_{10}) = 0.25$

- Five r_js $(r_2, r_4, r_6, r_8, r_{10})$ coincide directly with datapoint

Worked Example avg-11-pt prec: Query 1, interpolation



- Bold circles measured
- **thin circles interpolated**

Query 1			
Rank		R	P
1	X	.20	1.00
2			
3	X	.40	.67
4			
5			
6	X	.60	.50
7			
8			
9			
10	X	.80	.40
11			
12			
13			
14			
15			
16			
17			
18			
19			
20	X	1.00	.25

$\tilde{P}_1(r_0) = 1.00$

$\tilde{P}_1(r_1) = 1.00$

$\tilde{P}_1(r_2) = 1.00$

$\tilde{P}_1(r_3) = .67$

$\tilde{P}_1(r_4) = .67$

$\tilde{P}_1(r_5) = .50$

$\tilde{P}_1(r_6) = .50$

$\tilde{P}_1(r_7) = .40$

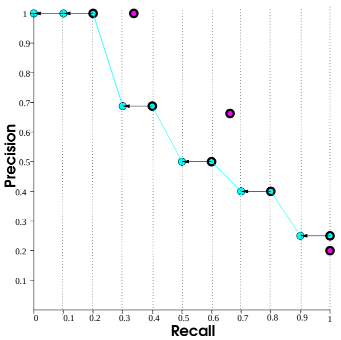
$\tilde{P}_1(r_8) = .40$

$\tilde{P}_1(r_9) = .25$

$\tilde{P}_1(r_{10}) = .25$

- The six other r_js (r_0, r_1, r_3, r_5, r_7, r_9) are interpolated.

Worked Example avg-11-pt prec: Query 2, measured data points



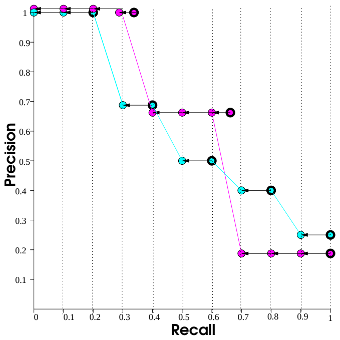
- Blue: Query 1; **Red: Query 2**
- **Bold circles measured**; thin circles interpol.

Query 2			
Rank	Relev.	R	P
1	X	.33	1.00
2			
3	X	.67	.67
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15	X	1.0	.2

$\tilde{P}_2(r_{10}) = .20$

- Only r_{10} coincides with a measured data point

Worked Example avg-11-pt prec: Query 2, interpolation



- Blue: Query 1; **Red: Query 2**
- Bold circles measured; **thin circles interpol.**

Query 2			
Rank	Relev.	R	P
1	X	.33	1.00
2			
3	X	.67	.67
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15	X	1.0	.2

$\tilde{P}_2(r_0) = 1.00$

$\tilde{P}_2(r_1) = 1.00$

$\tilde{P}_2(r_2) = 1.00$

$\tilde{P}_2(r_3) = 1.00$

$\tilde{P}_2(r_4) = .67$

$\tilde{P}_2(r_5) = .67$

$\tilde{P}_2(r_6) = .67$

$\tilde{P}_2(r_7) = .20$

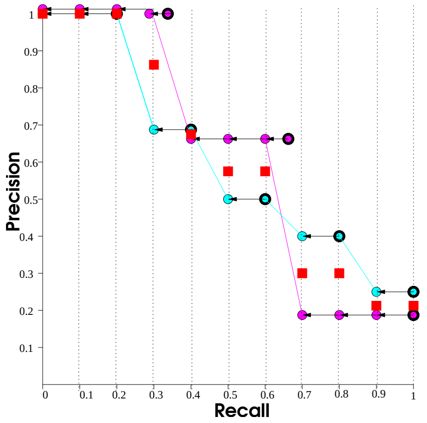
$\tilde{P}_2(r_8) = .20$

$\tilde{P}_2(r_9) = .20$

$\tilde{P}_2(r_{10}) = .20$

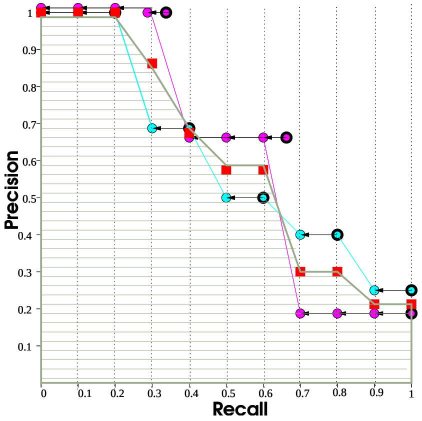
- 10 of the r_js are interpolated

Worked Example avg-11-pt prec: averaging



- Now average at each p_j
- over N (number of queries)
- → 11 averages

Worked Example avg-11-pt prec: area/result



- End result:
- 11 point average precision
- Approximation of area under prec. recall curve

Lecture 7: Relevance Feedback and Query Expansion

Information Retrieval
Computer Science Tripos Part II

Ronan Cummins

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

2017

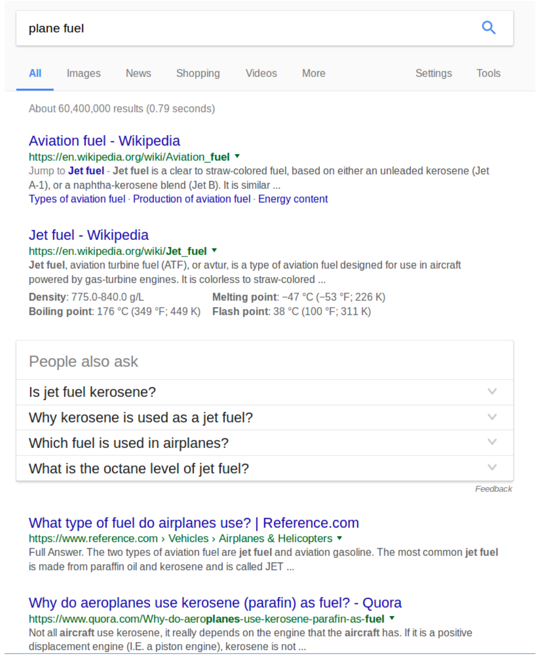
Overview

1. Introduction
2. Relevance Feedback
   - Rocchio Algorithm
   - Relevance-Based Language Models
3. Query Expansion

Motivation

- The same word can have different meanings (polysemy)
- Two different words can have the same meaning (synonymy)
- Vocabulary of searcher may not match that of the documents
- Consider the query = $\{plane\ fuel\}$
- While this is relatively unambiguous (wrt the meaning of each word in context), exact matching will miss documents containing *aircraft*, *airplane*, or *jet*
- Relevance feedback and query expansion aim to overcome the problem of *synonymy*

Example



plane fuel

About 60,400,000 results (0.79 seconds)

Aviation fuel - Wikipedia
https://en.wikipedia.org/wiki/Aviation_**fuel** ▾
Jump to **Jet fuel** - **Jet fuel** is a clear to straw-colored fuel, based on either an unleaded kerosene (Jet A-1), or a naphtha-kerosene blend (Jet B). It is similar ...
Types of aviation fuel · Production of aviation fuel · Energy content

Jet fuel - Wikipedia
https://en.wikipedia.org/wiki/**Jet_fuel** ▾
Jet fuel, aviation turbine fuel (ATF), or avtur, is a type of aviation fuel designed for use in aircraft powered by gas-turbine engines. It is colorless to straw-colored ...

Density: 775.0-840.0 g/L	**Melting point**: −47 °C (−53 °F; 226 K)
Boiling point: 176 °C (349 °F; 449 K)	**Flash point**: 38 °C (100 °F; 311 K)

People also ask

Is jet fuel kerosene?

Why kerosene is used as a jet fuel?

Which fuel is used in airplanes?

What is the octane level of jet fuel?

Feedback

What type of fuel do airplanes use? | Reference.com
https://www.reference.com › Vehicles › Airplanes & Helicopters ▾
Full Answer. The two types of aviation fuel are **jet fuel** and aviation gasoline. The most common **jet fuel** is made from paraffin oil and kerosene and is called JET ...

Why do aeroplanes use kerosene (parafin) as fuel? - Quora
https://www.quora.com/Why-do-aero**planes**-use-kerosene-parafin-as-**fuel** ▾
Not all **aircraft** use kerosene, it really depends on the engine that the **aircraft** has. If it is a positive displacement engine (I.E. a piston engine), kerosene is not ...

Improving Recall

- Local analysis: query-time analysis on a portion of documents returned for a user query
  - Main local method: relevance feedback
- Global analysis: perform a global analysis once (e.g., of collection) to produce thesaurus
  - Use thesaurus for query expansion

Overview

1 Introduction

2 Relevance Feedback
- Rocchio Algorithm
- Relevance-Based Language Models

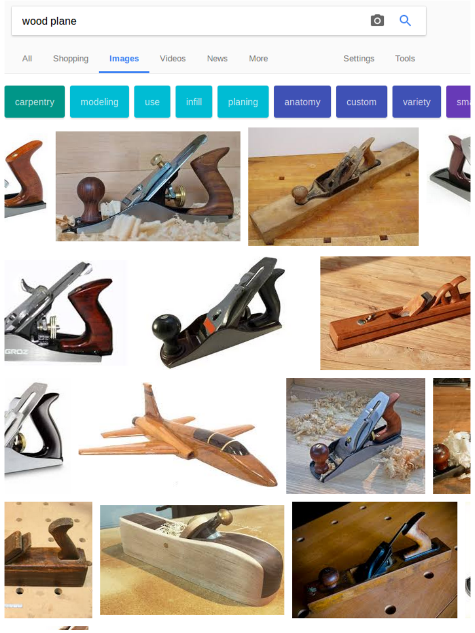
3 Query Expansion

The Basics

- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant (possibly some as non-relevant).
- Search engine computes a new representation of the information need.
- Hope: better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall (and possibly also better precision).

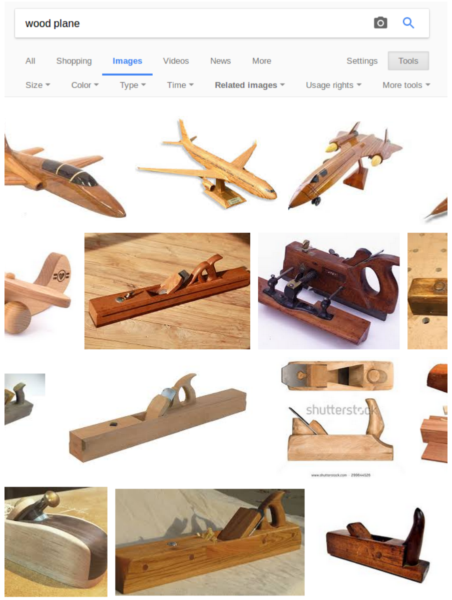
A limited form of RF is often expressed as “more like this” or ”find similar”.

Example



wood plane

Example



wood plane

Outline

1. Introduction
2. Relevance Feedback
   - Rocchio Algorithm
   - Relevance-Based Language Models
3. Query Expansion

Rocchio Basics

- Developed in the late 60s or early 70s.
- It was developed using the VSM as its basis.
- Therefore, we represent documents as points in a high-dimensional term space.
- Uses centroids to calculate the center of a set of documents.

Rocchio Diagram

Rocchio aims to find the optimal query $\vec{q}_{opt}$ that maximises:

$$\vec{q}_{opt} = \arg\max_{\vec{q}} [sim(\vec{q}, C_r) - sim(\vec{q}, C_{nr})] \quad (1)$$

where $sim(\vec{q}, C_r)$ is the similarity between a query q and the set of relevant documents C_r.

Rocchio Diagram

Rocchio aims to find the optimal query $\vec{q}_{opt}$ that maximises:

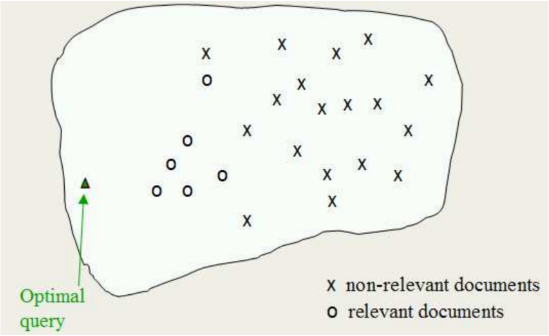
$$\vec{q}_{opt} = \arg\max_{\vec{q}} [sim(\vec{q}, C_r) - sim(\vec{q}, C_{nr})] \quad (1)$$

where $sim(\vec{q}, C_r)$ is the similarity between a query q and the set of relevant documents C_r. Using cosine similarity the optimal query becomes:

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{|C_{nr}|} \sum_{\vec{d}_j \in C_{nr}} \vec{d}_j \quad (2)$$

which is the difference between the centroids of the relevant and non-relevant document vectors.

Rocchio Diagram



Rocchio in practice

- However, we usually do not know the full relevant and non-relevant sets.
- For example, a user might only label a few documents as relevant.

Therefore, in practice Rocchio is often parameterised as follows:

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \gamma \frac{1}{|C_{nr}|} \sum_{\vec{d}_j \in C_{nr}} \vec{d}_j \quad (3)$$

where α, β, and γ are weights that are attached to each component.

Rocchio Summary

- Rocchio has been shown useful for increasing recall
- Contains aspects of positive and negative feedback
- Positive feedback is much more valuable (i.e. indications of what *is* relevant and $\gamma < \beta$
- Reasonable values of the parameters are $\alpha = 1.0$, $\beta = 0.75$, $\gamma = 0.15$

Outline

1. Introduction
2. Relevance Feedback
   - Rocchio Algorithm
   - Relevance-Based Language Models
3. Query Expansion

Relevance-Based Language Models I

- The query-likelihood language model (earlier lecture) had no concept of relevance (if you remember)
- Relevance-Based language models take a probabilistic language modelling approach to modelling relevance
- The main assumption is that a document is generated from either one of two classes (i.e. relevant or non-relevant)
- Documents are then ranked according to their probability of being drawn from the relevance class

$$P(R|D) = \frac{P(D|R)P(R)}{P(D|R)P(R) + P(D|NR)P(NR)} \tag{4}$$

which is rank equivalent to ranking by log-odds

$$= log \frac{P(D|R)}{P(D|NR)} \tag{5}$$

Relevance-Based Language Models II

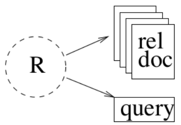


Figure 1: Queries and relevant documents are random samples from an underlying relevance model *R*. Note: the sampling process could be different for queries and documents.

- Lavrenko (2001) introduced the idea of relevance-based language models
- Outlined a number of different generative models
- One of the best performing models is one called RM3 (useful for both relevance and pseudo-relevance feedback)

Relevance-Based Language Models III

- Given a set of known relevant documents R one can estimate a relevance language model (e.g. multinomial θ_R)
- In practice, this can be smoothed with the original query model and a background model (not shown)

One could estimate the relevance model as:

$$(1 - \pi)\theta_R + \pi\theta_q \tag{6}$$

where π controls how much of the original query one wishes to retain.

Problems?

- Relevance feedback is expensive
- Relevance feedback creates long modified queries
- Long queries are expensive to process
- Users are reluctant to provide explicit feedback
- Its often hard to understand why a particular document was retrieved after applying relevance feedback

When does RF work?

- When users are willing to give feedback!
- When the user knows the terms in the collection well enough for an initial query.
- When relevant documents contain similar terms (similar to the cluster hypothesis)

> *The cluster hypothesis states that if there is a document from a cluster that is relevant to a search request, then it is likely that other documents from the same cluster are also relevant. - Jardine and van Rijsbergen*

Relevance Feedback - Evaluation

- How to evaluate if RF works?

Relevance Feedback - Evaluation

- How to evaluate if RF works?
- Have two collections, with relevance judgements for the same information needs (queries)
- User studies: time taken to find # of relevant documents (with and without feedback)

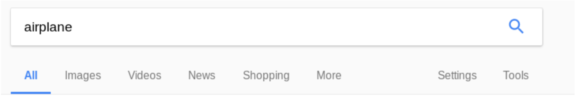
Other types of relevance feedback

- Implicit relevance feedback
- Pseudo relevance feedback - when does it work?

Overview

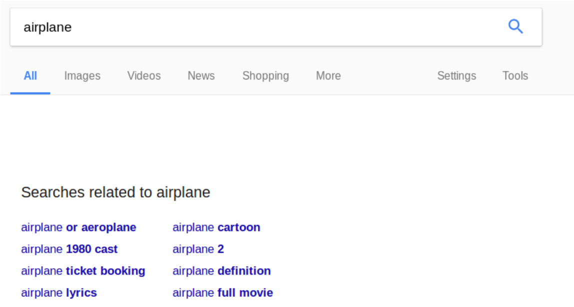
1. Introduction
2. Relevance Feedback
   - Rocchio Algorithm
   - Relevance-Based Language Models
3. Query Expansion

Query Expansion Motivation



airplane

Query Expansion Motivation



airplane

All Images Videos News Shopping More Settings Tools

Searches related to airplane

airplane **or aeroplane**	airplane **cartoon**
airplane **1980 cast**	airplane **2**
airplane **ticket booking**	airplane **definition**
airplane **lyrics**	airplane **full movie**

Query Expansion Introduction

- Query expansion is another method for increasing recall
- We use “global query expansion” to refer to “global methods for query reformulation”
- In global query expansion, the query is modified based on some global resource, i.e. a resource that is not query-dependent
- Often the problem aims to find (near-)synonyms
- Distributional Semantics (word embeddings)
- What’s the different between “local” and “global” methods?

Query Expansion Methods

- Use of a controlled vocabulary that is maintained by human editors (e.g. sets of keywords for publications - Medline)
- A manual thesaurus (e.g. wordnet)
- An automatically derived thesaurus
- Query reformulations based on query log mining (i.e. what the large search engines do)

Automatic thesaurus generation I

- Let A be a term-document matrix
- Where each cell A_{td} is a weighted count of term t in document (or window) d
- Row normalise the matrix (e.g. L2 normalisation)
- Then $C = AA^T$ is a term-term similarity matrix
- The similarity between any two terms u and v is in C_{uv}
- Given any particular term q, the most similar terms can be easily retrieved

Automatic thesaurus generation II

- Other approaches involve distributional semantics
- Where words with similar meanings appear in similar contexts
- Word embeddings - word2vec, glove, etc
- Can be useful but global expansion still suffers from problems of polysemy
- A naive approach to word-level expansion might lead to {apple computer} → {apple fruit computer}

Summary

- QE is transparent in that it allows the user to see (select) expansion terms
- Local approaches (PRF) to expanding queries tend to be more effective
- E.g. {apple computer} → {apple computer jobs iphone ipad macintosh}
- Local approaches tend to automatically disambiguate the individual query terms. Why?
- Query log mining approaches have also been shown to be useful

Reading

- Manning, Raghavan, Schütze: Introduction to Information Retrieval (MRS), chapter 9: Relevance feedback and query expansion, chapter 16.1: Clustering in information retrieval
- Victor Lavrenko and W. Bruce Croft: Relevance-Based Language Models

Lecture 8: Linkage algorithms and web search

Information Retrieval
Computer Science Tripos Part II

Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE
ronan.cummins@cl.cam.ac.uk

2017

[1] Adapted from Simone Teufel's original slides

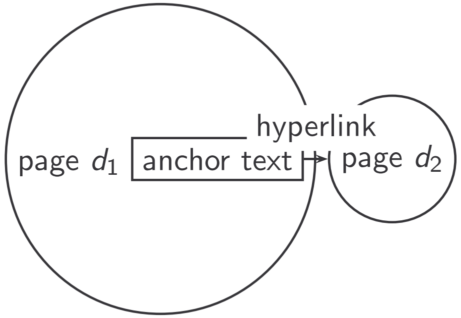
Upcoming today

- Anchor text: What exactly are links on the web and why are they important for IR?
- PageRank: the original algorithm that was used for link-based ranking on the web
- Hubs & Authorities: an alternative link-based ranking algorithm

Overview

1. Anchor text
2. PageRank
3. Wrap up

The web as a directed graph

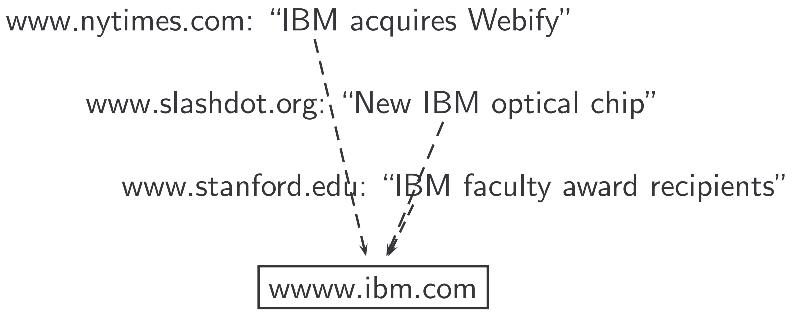


- Assumption 1: A hyperlink is a quality signal.
  - The hyperlink $d_1 \to d_2$ indicates that d_1's author deems d_2 high-quality and relevant.
- Assumption 2: The anchor text describes the content of d_2.
  - We use anchor text somewhat loosely here for: the text surrounding the hyperlink.
  - Example: "You can find cheap cars <a href=http://...>here</a>."
  - Anchor text: "You can find cheap cars here"

[text of d_2] only vs. [text of d_2] + [anchor text $\to d_2$]

- Searching on [text of d_2] + [anchor text $\to d_2$] is often more effective than searching on [text of d_2] only.
- Example: Query *IBM*
  - Matches IBM's copyright page
  - Matches many spam pages
  - Matches IBM wikipedia article
  - May not match IBM home page!
  - ...if IBM home page is mostly graphics
- Searching on [anchor text $\to d_2$] is better for the query *IBM*.
  - In this representation, the page with the most occurrences of *IBM* is www.ibm.com.

Anchor text containing *IBM* pointing to www.ibm.com


www.nytimes.com: "IBM acquires Webify"

www.slashdot.org: "New IBM optical chip"

www.stanford.edu: "IBM faculty award recipients"

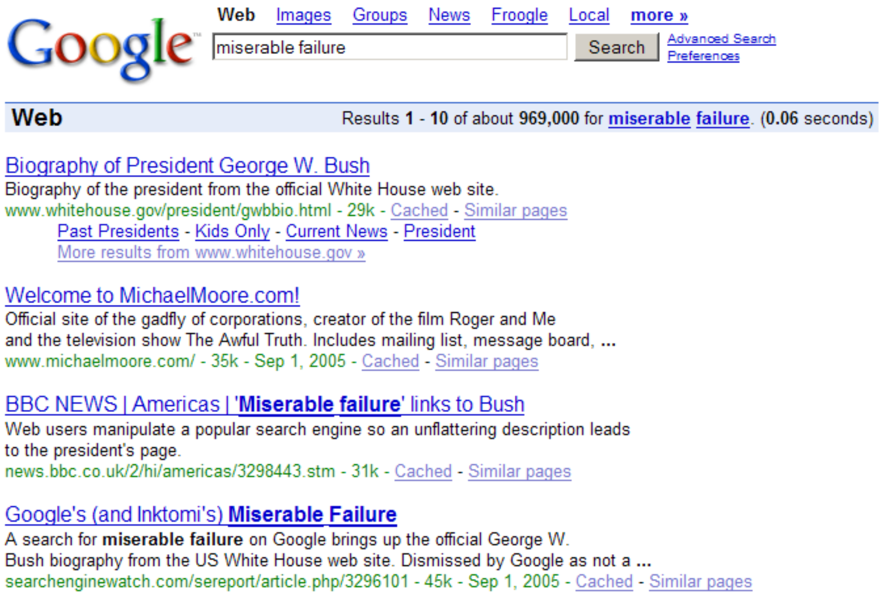
wwww.ibm.com

- Thus: Anchor text is often a better description of a page's content than the page itself.
- Anchor text can be weighted more highly than document text. (based on Assumptions 1&2)

Google bombs

- A Google bomb is a search with "bad" results due to maliciously manipulated anchor text.
- Google introduced a new weighting function in 2007 that fixed many Google bombs.
- Still some remnants: [dangerous cult] on Google, Bing, Yahoo
  - Coordinated link creation by those who dislike the Church of Scientology
- Defused Google bombs: [dumb motherf....], [who is a failure?], [evil empire]

A historic google bomb



miserable failure

Web Results **1** - **10** of about **969,000** for **miserable failure**. (**0.06** seconds)

Biography of President George W. Bush
Biography of the president from the official White House web site.
www.whitehouse.gov/president/gwbbio.html - 29k - Cached - Similar pages
Past Presidents - Kids Only - Current News - President
More results from www.whitehouse.gov »

Welcome to MichaelMoore.com!
Official site of the gadfly of corporations, creator of the film Roger and Me and the television show The Awful Truth. Includes mailing list, message board, ...
www.michaelmoore.com/ - 35k - Sep 1, 2005 - Cached - Similar pages

BBC NEWS | Americas | '**Miserable failure**' links to Bush
Web users manipulate a popular search engine so an unflattering description leads to the president's page.
news.bbc.co.uk/2/hi/americas/3298443.stm - 31k - Cached - Similar pages

Google's (and Inktomi's) **Miserable Failure**
A search for **miserable failure** on Google brings up the official George W. Bush biography from the US White House web site. Dismissed by Google as not a ...
searchenginewatch.com/sereport/article.php/3296101 - 45k - Sep 1, 2005 - Cached - Similar pages

Origins of PageRank: Citation Analysis

- We can use the same formal representation (as DAG) for
  - citations in the scientific literature
  - hyperlinks on the web
- Appropriately weighted citation frequency is an excellent measure of quality . . .
  - . . . both for web pages and for scientific publications.
- Next: PageRank algorithm for computing weighted citation frequency on the web

Overview

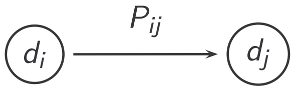
1. Anchor text
2. PageRank
3. Wrap up

Model behind PageRank: Random walk

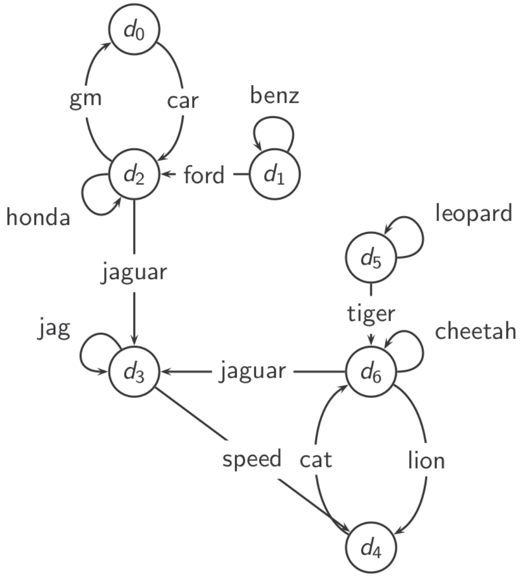
- Imagine a web surfer doing a random walk on the web
  - Start at a random page
  - At each step, go out of the current page along one of the links on that page, equiprobably
- In the steady state, each page has a long-term visit rate.
- This long-term visit rate is the page's PageRank.
- PageRank = long-term visit rate = steady state probability

Formalisation of random walk: Markov chains

- A Markov chain consists of N states, plus an $N \times N$ transition probability matrix P.
- state = page
- At each step, we are on exactly one of the pages.
- For $1 \leq i, j \leq N$, the matrix entry P_{ij} tells us the probability of j being the next page, given we are currently on page i.
- Clearly, for all i, $\sum_{j=1}^{N} P_{ij} = 1$



Example web graph



Link matrix for example

	d_0	d_1	d_2	d_3	d_4	d_5	d_6
d_0	0	0	1	0	0	0	0
d_1	0	1	1	0	0	0	0
d_2	1	0	1	1	0	0	0
d_3	0	0	0	1	1	0	0
d_4	0	0	0	0	0	0	1
d_5	0	0	0	0	0	1	1
d_6	0	0	0	1	1	0	1

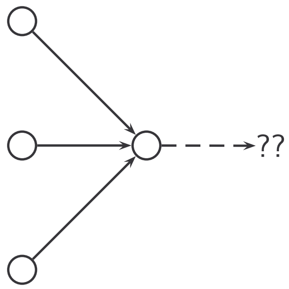
Transition probability matrix P for example

	d_0	d_1	d_2	d_3	d_4	d_5	d_6
d_0	0.00	0.00	1.00	0.00	0.00	0.00	0.00
d_1	0.00	0.50	0.50	0.00	0.00	0.00	0.00
d_2	0.33	0.00	0.33	0.33	0.00	0.00	0.00
d_3	0.00	0.00	0.00	0.50	0.50	0.00	0.00
d_4	0.00	0.00	0.00	0.00	0.00	0.00	1.00
d_5	0.00	0.00	0.00	0.00	0.00	0.50	0.50
d_6	0.00	0.00	0.00	0.33	0.33	0.00	0.33

Long-term visit rate

- Recall: PageRank = long-term visit rate
- Long-term visit rate of page d is the probability that a web surfer is at page d at a given point in time.
- Next: what properties must hold of the web graph for the long-term visit rate to be well defined?
- The web graph must correspond to an ergodic Markov chain.
- First a special case: The web graph must not contain dead ends.

Dead ends



- The web is full of dead ends.
- Random walk can get stuck in dead ends.
- If there are dead ends, long-term visit rates are not well-defined (or non-sensical).

Teleporting – to get us out of dead ends

- At a dead end, jump to a random web page with prob. $1/N$.
- At a non-dead end, with probability 10%, jump to a random web page (to each with a probability of $0.1/N$).
- With remaining probability (90%), follow a random hyperlink on the page.
  - For example, if the page has 4 outgoing links: randomly choose one with probability (1-0.10)/4=0.225
- 10% is a parameter, the teleportation rate.
- Note: "jumping" from dead end is independent of teleportation rate.

Teleporting – formula

$$P' = (1 - \alpha) \cdot P + \alpha \cdot T \qquad (1)$$

where T is the teleportation matrix and P is a stochastic matrix

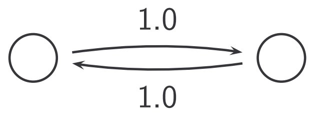
- what is T?
- An $N \times N$ matrix full of $1/N$
- α is the probability of teleporting

Result of teleporting

- With teleporting, we cannot get stuck in a dead end.
- But even without dead ends, a graph may not have well-defined long-term visit rates.
- More generally, we require that the Markov chain be ergodic.

Ergodic Markov chains

- A Markov chain is ergodic iff it is irreducible and aperiodic.
- Irreducibility. Roughly: there is a path from any page to any other page.
- Aperiodicity. Roughly: The pages cannot be partitioned such that the random walker visits the partitions sequentially.
- A non-ergodic Markov chain:



Ergodic Markov chains

- Theorem: For any ergodic Markov chain, there is a unique long-term visit rate for each state.
- This is the steady-state probability distribution.
- Over a long time period, we visit each state in proportion to this rate.
- It doesn't matter where we start.
- Teleporting makes the web graph ergodic.
- ⇒ Web-graph+teleporting has a steady-state probability distribution.
- ⇒ Each page in the web-graph+teleporting has a PageRank.

Where we are

- We now know what to do to make sure we have a well-defined PageRank for each page.
- Next: how to compute PageRank

Formalization of "visit": Probability vector

- A probability (row) vector $\vec{x} = (x_1, \ldots, x_N)$ tells us where the random walk is at any point.
- Example: $\begin{array}{cccccccccccc} (& 0 & 0 & 0 & \ldots & 1 & \ldots & 0 & 0 & 0 &) \\ & 1 & 2 & 3 & \ldots & i & \ldots & \text{N-2} & \text{N-1} & \text{N} & \end{array}$
- More generally: the random walk is on page i with probability x_i.
- Example:

$$\begin{array}{cccccccccccc} (& 0.05 & 0.01 & 0.0 & \ldots & 0.2 & \ldots & 0.01 & 0.05 & 0.03 &) \\ & 1 & 2 & 3 & \ldots & i & \ldots & \text{N-2} & \text{N-1} & \text{N} & \end{array}$$

- $\sum x_i = 1$

Change in probability vector

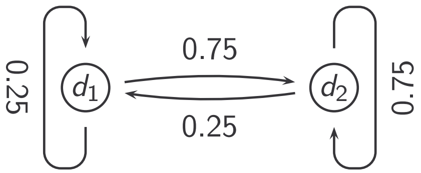
- If the probability vector is $\vec{x} = (x_1, \ldots, x_N)$ at this step, what is it at the next step?
- Recall that row i of the transition probability matrix P tells us where we go next from state i.
- So from $\vec{x}$, our next state is distributed as $\vec{x}P$.

Steady state in vector notation

- The steady state in vector notation is simply a vector $\vec{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)$ of probabilities.
- (We use $\vec{\pi}$ to distinguish it from the notation for the probability vector $\vec{x}$.)
- π_i is the long-term visit rate (or PageRank) of page i.
- So we can think of PageRank as a very long vector – one entry per page.

Steady-state distribution: Example

What is the PageRank / steady state in this example?



Steady-state distribution: Example

	x_1 $P_t(d_1)$	x_2 $P_t(d_2)$		
			$P_{11} = 0.25$ $P_{21} = 0.25$	$P_{12} = 0.75$ $P_{22} = 0.75$
t_0	0.25	0.75		
t_1	0.25	0.75	(convergence)	

$$\begin{aligned} P_t(d_1) &= P_{t-1}(d_1) \cdot P_{11} + P_{t-1}(d_2) \cdot P_{21} \\ &\quad 0.25 \cdot 0.25 + 0.75 \cdot 0.25 = 0.25 \\ P_t(d_2) &= P_{t-1}(d_1) \cdot P_{12} + P_{t-1}(d_2) \cdot P_{22} \\ &\quad 0.75 \cdot 0.25 + 0.75 \cdot 0.75 = 0.75 \end{aligned}$$

PageRank vector $= \vec{\pi} = (\pi_1, \pi_2) = (0.25, 0.75)$

How do we compute the steady state vector?

- In other words: how do we compute PageRank?
- Recall: $\vec{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)$ is the PageRank vector, the vector of steady-state probabilities ...
- ...and if the distribution in this step is $\vec{x}$, then the distribution in the next step is $\vec{x}P$.
- But $\vec{\pi}$ is the steady state!
- So: $\vec{\pi} = \vec{\pi}P$
- Solving this matrix equation gives us $\vec{\pi}$.
- $\vec{\pi}$ is the principal left eigenvector for P ...
- ...that is, $\vec{\pi}$ is the left eigenvector with the largest eigenvalue.
- All transition probability matrices have largest eigenvalue 1.

One way of computing the PageRank $\vec{\pi}$

- Start with any distribution $\vec{x}$, e.g., uniform distribution
- After one step, we're at $\vec{x}P$.
- After two steps, we're at $\vec{x}P^2$.
- After k steps, we're at $\vec{x}P^k$.
- Algorithm: multiply $\vec{x}$ by increasing powers of P until convergence.
- This is called the power method.
- Recall: regardless of where we start, we eventually reach the steady state $\vec{\pi}$.
- Thus: we will eventually (in asymptotia) reach the steady state.

Computing PageRank: Power method

	x_1 $P_t(d_1)$	x_2 $P_t(d_2)$			
			$P_{11} = 0.1$ $P_{21} = 0.3$	$P_{12} = 0.9$ $P_{22} = 0.7$	
t_0	0	1	0.3	0.7	$= \vec{x}P$
t_1	0.3	0.7	0.24	0.76	$= \vec{x}P^2$
t_2	0.24	0.76	0.252	0.748	$= \vec{x}P^3$
t_3	0.252	0.748	0.2496	0.7504	$= \vec{x}P^4$
			. . .		. . .
t_∞	0.25	0.75	0.25	0.75	$= \vec{x}P^\infty$

PageRank vector $= \vec{\pi} = (\pi_1, \pi_2) = (0.25, 0.75)$

$$P_t(d_1) = P_{t-1}(d_1) * P_{11} + P_{t-1}(d_2) * P_{21}$$
$$P_t(d_2) = P_{t-1}(d_1) * P_{12} + P_{t-1}(d_2) * P_{22}$$

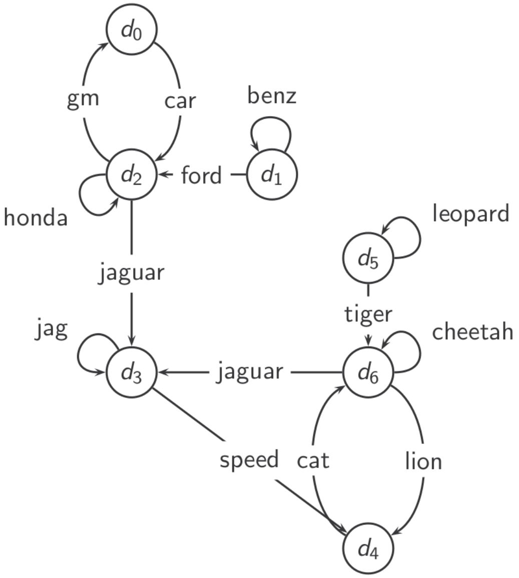
PageRank summary

- Preprocessing
  - Given graph of links, build initial matrix P
  - Ensure all rows sum to 1.0 to update P (for nodes with no outgoing links use $1/N$ for each element)
  - Apply teleportation with parameter α
  - From modified matrix, compute $\vec{\pi}$
  - $\vec{\pi}_i$ is the PageRank of page i.
- Query processing
  - Retrieve pages satisfying the query
  - Rank them by their PageRank (or at least a combination of PageRank and the relevance score)
  - Return reranked list to the user

PageRank issues

- Real surfers are not random surfers.
  - Examples of non-random surfing: back button, short vs. long paths, bookmarks, directories – and search!
  - $\rightarrow$ Markov model is not a good model of surfing.
  - But it's good enough as a model for our purposes.
- Simple PageRank ranking (as described on previous slide) produces bad results for many pages.
  - Consider the query [video service]
  - The Yahoo home page (i) has a very high PageRank and (ii) contains both *video* and *service*.
  - If we rank all Boolean hits according to PageRank, then the Yahoo home page would be top-ranked.
  - Clearly not desirable
- In practice: rank according to weighted combination of raw text match, anchor text match, PageRank & other factors

Example web graph



Transition (probability) matrix

	d_0	d_1	d_2	d_3	d_4	d_5	d_6
d_0	0.00	0.00	1.00	0.00	0.00	0.00	0.00
d_1	0.00	0.50	0.50	0.00	0.00	0.00	0.00
d_2	0.33	0.00	0.33	0.33	0.00	0.00	0.00
d_3	0.00	0.00	0.00	0.50	0.50	0.00	0.00
d_4	0.00	0.00	0.00	0.00	0.00	0.00	1.00
d_5	0.00	0.00	0.00	0.00	0.00	0.50	0.50
d_6	0.00	0.00	0.00	0.33	0.33	0.00	0.33

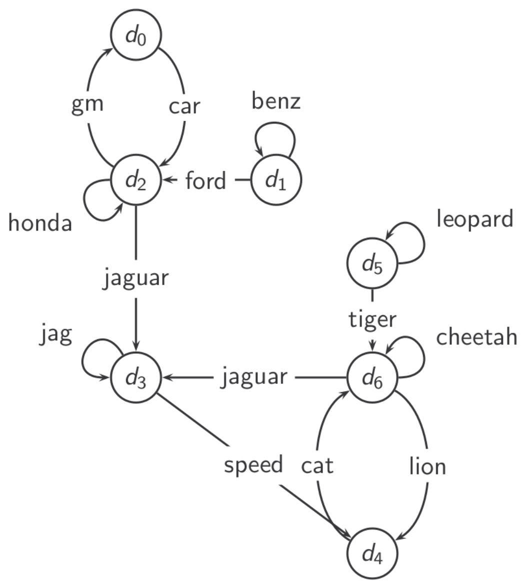
Transition matrix with teleporting ($\alpha = 0.14$)

	d_0	d_1	d_2	d_3	d_4	d_5	d_6
d_0	0.02	0.02	0.88	0.02	0.02	0.02	0.02
d_1	0.02	0.45	0.45	0.02	0.02	0.02	0.02
d_2	0.31	0.02	0.31	0.31	0.02	0.02	0.02
d_3	0.02	0.02	0.02	0.45	0.45	0.02	0.02
d_4	0.02	0.02	0.02	0.02	0.02	0.02	0.88
d_5	0.02	0.02	0.02	0.02	0.02	0.45	0.45
d_6	0.02	0.02	0.02	0.31	0.31	0.02	0.31

Power method vectors $\vec{x}P^k$

	$\vec{x}$	$\vec{x}P^1$	$\vec{x}P^2$	$\vec{x}P^3$	$\vec{x}P^4$	$\vec{x}P^5$	$\vec{x}P^6$	$\vec{x}P^7$	$\vec{x}P^8$	$\vec{x}P^9$	$\vec{x}P^{10}$	$\vec{x}P^{11}$	$\vec{x}P^{12}$	$\vec{x}P^{13}$
d_0	0.14	0.06	0.09	0.07	0.07	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05	0.05
d_1	0.14	0.08	0.06	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
d_2	0.14	0.25	0.18	0.17	0.15	0.14	0.13	0.12	0.12	0.12	0.12	0.11	0.11	0.11
d_3	0.14	0.16	0.23	0.24	0.24	0.24	0.24	0.25	0.25	0.25	0.25	0.25	0.25	0.25
d_4	0.14	0.12	0.16	0.19	0.19	0.20	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.21
d_5	0.14	0.08	0.06	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
d_6	0.14	0.25	0.23	0.25	0.27	0.28	0.29	0.29	0.30	0.30	0.30	0.30	0.31	0.31

Example web graph



How important is PageRank?

Frequent claim: PageRank is the most important component of web ranking. The reality:

- There are several components that are at least as important: e.g., anchor text, phrases, proximity, tiered indexes ...
- Rumour has it that PageRank in its original form (as presented here) now has a negligible impact on ranking
- However, variants of a page's PageRank are still an essential part of ranking.
- Google's official description of PageRank:

> "PageRank reflects our view of the importance of web pages by considering more than 500 million variables and 2 billion terms. Pages that we believe are important pages receive a higher PageRank and are more likely to appear at the top of the search results."

- Adressing link spam is difficult and crucial.

Overview

1. Anchor text
2. PageRank
3. Wrap up

Link Analysis

- PageRank is topic independent
- We also need to incorporate topicality (i.e. relevance)
- There is a version called Topic Sensitive PageRank
- And also Hyperlink-Induced Topic Search (HITS)

Take Home Messages

- Anchor text is a useful descriptor of the page it refers to
- Links can be used as another useful retrieval signal - one indicating authority
- PageRank can be viewed as the stationary distribution of a Markov chain
- Power iteration is *one simple method* of calculating the stationary distribution
- Topic sensitive variants exist

Reading

- MRS Chapter 21, excluding 21.3.3.