Introduction to Information Retrieval

http://informationretrieval.org

IIR 1: Boolean Retrieval

Hinrich Schütze, Christina Lioma

Institute for Natural Language Processing, University of Stuttgart

2010-04-26

Take-away

Take-away

- Administrativa

Take-away

- Administrativa
- Boolean Retrieval: Design and data structures of a simple information retrieval system

Take-away

- Administrativa
- Boolean Retrieval: Design and data structures of a simple information retrieval system
- What topics will be covered in this class?

Outline

Definition of *information retrieval*

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).

Definition of *information retrieval*

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).

Definition of *information retrieval*

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).

Definition of *information retrieval*

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).

Definition of *information retrieval*

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).

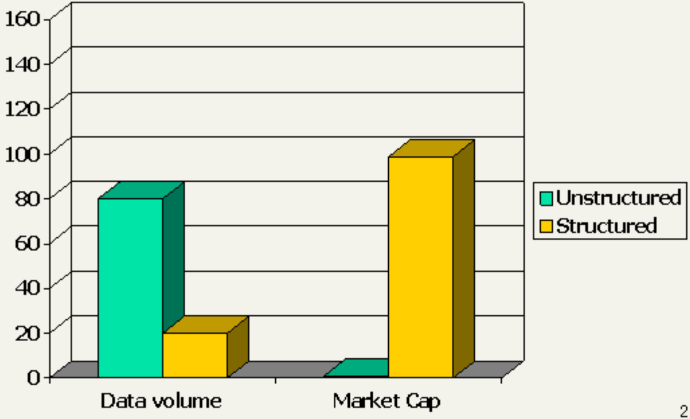
Definition of *information retrieval*

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).

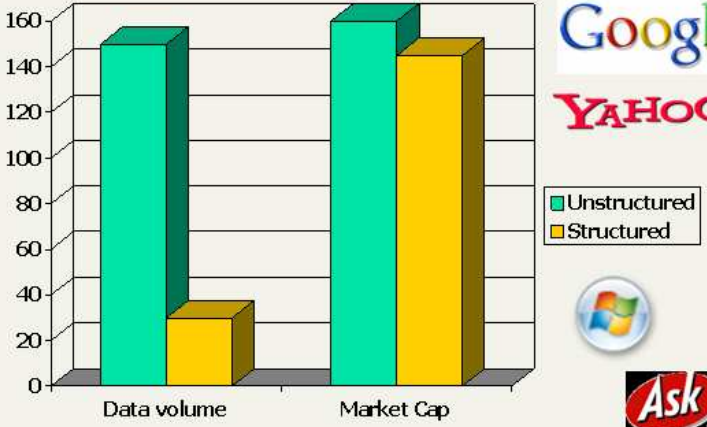
Definition of *information retrieval*

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).

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



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



Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.

Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS

Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The seach engine returns all documents that satisfy the Boolean expression.

Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The seach engine returns all documents that satisfy the Boolean expression.

Does Google use the Boolean model?

Outline

Unstructured data in 1650: Shakespeare

Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?

Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.

Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.
- Why is grep not the solution?

Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.
- Why is grep not the solution?
  - Slow (for large collections)

Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.
- Why is grep not the solution?
  - Slow (for large collections)
  - grep is line-oriented, IR is document-oriented

Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.
- Why is grep not the solution?
  - Slow (for large collections)
  - grep is line-oriented, IR is document-oriented
  - "NOT CALPURNIA" is non-trivial

Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.
- Why is grep not the solution?
  - Slow (for large collections)
  - grep is line-oriented, IR is document-oriented
  - "NOT CALPURNIA" is non-trivial
  - Other operations (e.g., find the word ROMANS near COUNTRYMAN) not feasible

Term-document 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	
. . .							

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.
Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Term-document 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	
...							

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.
Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Term-document 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	
...							

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.
Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:
  - Take the vectors for BRUTUS, CAESAR, and CALPURNIA
  - Complement the vector of CALPURNIA
  - Do a (bitwise) AND on the three vectors
  - 110100 AND 110111 AND 101111 = 100100

0/1 vector for Brutus

	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	
...							
result:	1	0	0	1	0	0	

Answers to query

Anthony and Cleopatra, Act III, Scene ii

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

Hamlet, Act III, Scene ii

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

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens
- $\Rightarrow$ total of 10^9 tokens

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens
- $\Rightarrow$ total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation $\Rightarrow$ size of document collection is about $6 \cdot 10^9 =$ 6 GB

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens
- $\Rightarrow$ total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation $\Rightarrow$ size of document collection is about $6 \cdot 10^9 =$ 6 GB
- Assume there are $M = 500{,}000$ distinct terms in the collection

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens
- $\Rightarrow$ total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation $\Rightarrow$ size of document collection is about $6 \cdot 10^9 =$ 6 GB
- Assume there are $M = 500{,}000$ distinct terms in the collection
- (Notice that we are making a term/token distinction.)

Can't build the incidence matrix

- $M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.

Can’t build the incidence matrix

- $M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.

Can't build the incidence matrix

- $M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.

Can't build the incidence matrix

- $M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
- What is a better representations?

Can't build the incidence matrix

- $M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
- What is a better representations?
  - We only record the 1s.

Inverted Index

For each term *t*, we store a list of all documents that contain *t*.

BRUTUS	$\longrightarrow$	1, 2, 4, 11, 31, 45, 173, 174
CAESAR	$\longrightarrow$	1, 2, 4, 5, 6, 16, 57, 132, ...
CALPURNIA	$\longrightarrow$	2, 31, 54, 101
⋮		
dictionary		**postings**

Inverted Index

For each term *t*, we store a list of all documents that contain *t*.

BRUTUS	$\longrightarrow$	1	2	4	11	31	45	173	174	
CAESAR	$\longrightarrow$	1	2	4	5	6	16	57	132	...
CALPURNIA	$\longrightarrow$	2	31	54	101					
⋮										
dictionary		**postings**								

Inverted Index

For each term *t*, we store a list of all documents that contain *t*.

BRUTUS	$\longrightarrow$	1	2	4	11	31	45	173	174	
CAESAR	$\longrightarrow$	1	2	4	5	6	16	57	132	...
CALPURNIA	$\longrightarrow$	2	31	54	101					
⋮										

dictionary **postings**

Inverted index construction

1. Collect the documents to be indexed:
   $\boxed{\text{Friends, Romans, countrymen.}}$ $\boxed{\text{So let it be with Caesar}}$ ...
2. Tokenize the text, turning each document into a list of tokens:
   $\boxed{\text{Friends}}$ $\boxed{\text{Romans}}$ $\boxed{\text{countrymen}}$ $\boxed{\text{So}}$ ...
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: $\boxed{\text{friend}}$ $\boxed{\text{roman}}$ $\boxed{\text{countryman}}$ $\boxed{\text{so}}$ ...
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Tokenization and preprocessing

Doc 1. I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.
Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

$\Longrightarrow$

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

Generate postings

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

$\Longrightarrow$

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

Sort postings

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

$\Longrightarrow$

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

Create postings lists, determine document frequency

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

$\Longrightarrow$

term	doc. freq.	→	postings lists
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

Split the result into dictionary and postings file

BRUTUS	⟶	1	2	4	11	31	45	173	174	
CAESAR	⟶	1	2	4	5	6	16	57	132	...
CALPURNIA	⟶	2	31	54	101					
⋮										

dictionary **postings file**

Later in this course

- Index construction: how can we create inverted indexes for large collections?

Later in this course

- Index construction: how can we create inverted indexes for large collections?
- How much space do we need for dictionary and index?

Later in this course

- Index construction: how can we create inverted indexes for large collections?
- How much space do we need for dictionary and index?
- Index compression: how can we efficiently store and process indexes for large collections?

Later in this course

- Index construction: how can we create inverted indexes for large collections?
- How much space do we need for dictionary and index?
- Index compression: how can we efficiently store and process indexes for large collections?
- Ranked retrieval: what does the inverted index look like when we want the "best" answer?

Outline

1. Introduction
2. Inverted index
3. Processing Boolean queries
4. Query optimization
5. Course overview

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary
  4. Retrieve its postings list from the postings file

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary
  4. Retrieve its postings list from the postings file
  5. Intersect the two postings lists

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary
  4. Retrieve its postings list from the postings file
  5. Intersect the two postings lists
  6. Return intersection to user

Intersecting two postings lists

$$\text{B}\textsc{rutus} \longrightarrow \boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$$

$$\text{C}\textsc{alpurnia} \longrightarrow \boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$$

Intersection $\Longrightarrow$

Intersecting two postings lists

BRUTUS	$\longrightarrow$	$\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$
CALPURNIA	$\longrightarrow$	$\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$
Intersection	$\Longrightarrow$	

Intersecting two postings lists

BRUTUS	$\longrightarrow$	$\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$
CALPURNIA	$\longrightarrow$	$\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$
Intersection	$\Longrightarrow$	

Intersecting two postings lists

BRUTUS	$\longrightarrow$	$\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$
CALPURNIA	$\longrightarrow$	$\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$
Intersection	$\Longrightarrow$	$\boxed{2}$

Intersecting two postings lists

BRUTUS $\longrightarrow$ $\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$

CALPURNIA $\longrightarrow$ $\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$

Intersection $\Longrightarrow$ $\boxed{2}$

Intersecting two postings lists

Brutus $\longrightarrow$ $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

Calpurnia $\longrightarrow$ $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection $\Longrightarrow$ $\boxed{2}$

Intersecting two postings lists

BRUTUS $\longrightarrow$ $\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$

CALPURNIA $\longrightarrow$ $\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$

Intersection $\Longrightarrow$ $\boxed{2}$

Intersecting two postings lists

BRUTUS	$\longrightarrow$	$\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$
CALPURNIA	$\longrightarrow$	$\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$
Intersection	$\Longrightarrow$	$\boxed{2} \to \boxed{31}$

Intersecting two postings lists

BRUTUS	$\longrightarrow$	$\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$
CALPURNIA	$\longrightarrow$	$\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$
Intersection	$\Longrightarrow$	$\boxed{2} \rightarrow \boxed{31}$

Intersecting two postings lists

BRUTUS ⟶ $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA ⟶ $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection ⟹ $\boxed{2} \rightarrow \boxed{31}$

Intersecting two postings lists

BRUTUS	$\longrightarrow$	$\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$
CALPURNIA	$\longrightarrow$	$\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$
Intersection	$\Longrightarrow$	$\boxed{2} \to \boxed{31}$

Intersecting two postings lists

BRUTUS $\longrightarrow$ $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA $\longrightarrow$ $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection $\Longrightarrow$ $\boxed{2} \rightarrow \boxed{31}$

Intersecting two postings lists

Brutus	$\longrightarrow$	$\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$
Calpurnia	$\longrightarrow$	$\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$
Intersection	$\Longrightarrow$	$\boxed{2} \to \boxed{31}$

- This is linear in the length of the postings lists.

Intersecting two postings lists

BRUTUS	$\longrightarrow$	$\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$
CALPURNIA	$\longrightarrow$	$\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$
Intersection	$\Longrightarrow$	$\boxed{2} \rightarrow \boxed{31}$

- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

Intersecting two postings lists

```
INTERSECT(p1, p2)
  answer ← ⟨ ⟩
  while p1 ≠ NIL and p2 ≠ NIL
  do if docID(p1) = docID(p2)
       then ADD(answer, docID(p1))
            p1 ← next(p1)
            p2 ← next(p2)
       else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
  return answer
```

Query processing: Exercise

FRANCE ⟶ 1 → 2 → 3 → 4 → 5 → 7 → 8 → 9 → 11 → 12 → 13 → 14 → 15

PARIS ⟶ 2 → 6 → 10 → 12 → 14

LEAR ⟶ 12 → 15

Compute hit list for ((paris AND NOT france) OR lear)

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
  - You know exactly what you are getting.

Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
    - Boolean queries are queries that use AND, OR and NOT to join query terms.
    - Views each document as a set of terms.
    - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
    - You know exactly what you are getting.
- Many search systems you use are also Boolean: spotlight, email, intranet etc.

Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers

Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data

Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.

Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.
- In 2005, Boolean search (called "Terms and Connectors" by Westlaw) was still the default, and used by a large percentage of users . . .

Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.
- In 2005, Boolean search (called "Terms and Connectors" by Westlaw) was still the default, and used by a large percentage of users . . .
- . . . although ranked retrieval has been available since 1992.

Westlaw: Example queries

Information need: Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company

Query: "trade secret" /s disclos! /s prevent /s employe!

Westlaw: Example queries

Information need: Requirements for disabled people to be able to access a workplace

Query: disab! /p access! /s work-site work-place (employment /3 place)

Westlaw: Example queries

Information need: Cases about a host's responsibility for drunk guests

Query: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Westlaw: Comments

- Proximity operators: /3 = within 3 words, /s = within a sentence, /p = within a paragraph

Westlaw: Comments

- Proximity operators: /3 = within 3 words, /s = within a sentence, /p = within a paragraph
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)

Westlaw: Comments

- Proximity operators: /3 = within 3 words, /s = within a sentence, /p = within a paragraph
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)
- Long, precise queries: incrementally developed, not like web search

Westlaw: Comments

- Proximity operators: /3 = within 3 words, /s = within a sentence, /p = within a paragraph
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)
- Long, precise queries: incrementally developed, not like web search
- Why professional searchers often like Boolean search: precision, transparency, control

Westlaw: Comments

- Proximity operators: /3 = within 3 words, /s = within a sentence, /p = within a paragraph
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)
- Long, precise queries: incrementally developed, not like web search
- Why professional searchers often like Boolean search: precision, transparency, control
- When are Boolean queries the best way of searching? Depends on: information need, searcher, document collection, . . .

Outline

1. Introduction
2. Inverted index
3. Processing Boolean queries
4. Query optimization
5. Course overview

Query optimization

- Consider a query that is an AND of n terms, $n > 2$

Query optimization

- Consider a query that is an AND of n terms, $n > 2$
- For each of the terms, get its postings list, then AND them together

Query optimization

- Consider a query that is an AND of n terms, $n > 2$
- For each of the terms, get its postings list, then AND them together
- Example query: BRUTUS AND CALPURNIA AND CAESAR

Query optimization

- Consider a query that is an AND of n terms, $n > 2$
- For each of the terms, get its postings list, then AND them together
- Example query: BRUTUS AND CALPURNIA AND CAESAR
- What is the best order for processing this query?

Query optimization

- Example query: Brutus AND Calpurnia AND Caesar

Query optimization

- Example query: Brutus AND Calpurnia AND Caesar
- Simple and effective optimization: Process in order of increasing frequency

Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR
- Simple and effective optimization: Process in order of increasing frequency
- Start with the shortest postings list, then keep cutting further

Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR
- Simple and effective optimization: Process in order of increasing frequency
- Start with the shortest postings list, then keep cutting further

BRUTUS	⟶	1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
CALPURNIA	⟶	2 → 31 → 54 → 101
CAESAR	⟶	5 → 31

Query optimization

- Example query: Brutus AND Calpurnia AND Caesar
- Simple and effective optimization: Process in order of increasing frequency
- Start with the shortest postings list, then keep cutting further
- In this example, first Caesar, then Calpurnia, then Brutus

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

Calpurnia ⟶ 2 → 31 → 54 → 101

Caesar ⟶ 5 → 31

Optimized intersection algorithm for conjunctive queries

```
INTERSECT(⟨t_1, ..., t_n⟩)
    terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
    result ← postings(first(terms))
    terms ← rest(terms)
    while terms ≠ NIL and result ≠ NIL
    do result ← INTERSECT(result, postings(first(terms)))
        terms ← rest(terms)
    return result
```

More general optimization

- Example query: (MADDING OR CROWD) AND (IGNOBLE OR STRIFE)

More general optimization

- Example query: (MADDING OR CROWD) AND (IGNOBLE OR STRIFE)
- Get frequencies for all terms

More general optimization

- Example query: (MADDING OR CROWD) AND (IGNOBLE OR STRIFE)
- Get frequencies for all terms
- Estimate the size of each OR by the sum of its frequencies (conservative)

More general optimization

- Example query: (MADDING OR CROWD) AND (IGNOBLE OR STRIFE)
- Get frequencies for all terms
- Estimate the size of each OR by the sum of its frequencies (conservative)
- Process in increasing order of OR sizes

Outline

1. Introduction
2. Inverted index
3. Processing Boolean queries
4. Query optimization
5. Course overview

Course overview

- We are done with Chapter 1 of IIR (IIR 01).

Course overview

- We are done with Chapter 1 of IIR (IIR 01).
- Plan for the rest of the semester: 18–20 of the 21 chapters of IIR

Course overview

- We are done with Chapter 1 of IIR (IIR 01).
- Plan for the rest of the semester: 18–20 of the 21 chapters of IIR
- In what follows: teasers for most chapters – to give you a sense of what will be covered.

IIR 02: The term vocabulary and postings lists

- Phrase queries: "STANFORD UNIVERSITY"
- Proximity queries: GATES NEAR MICROSOFT
- We need an index that captures position information for phrase queries and proximity queries.

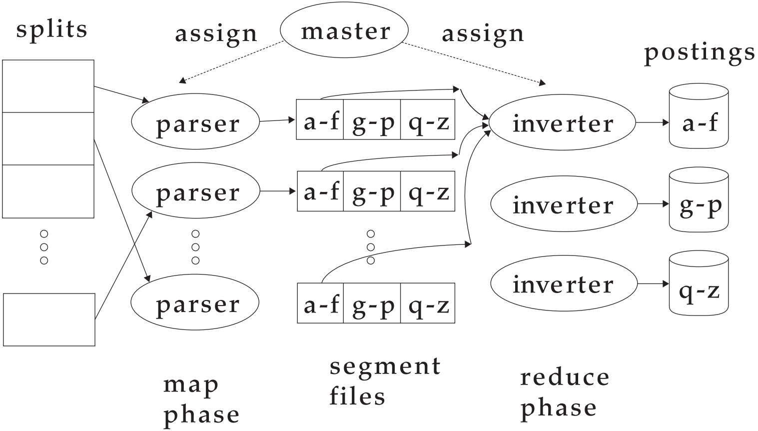
IIR 03: Dictionaries and tolerant retrieval

BO → aboard → about → boardroom → border

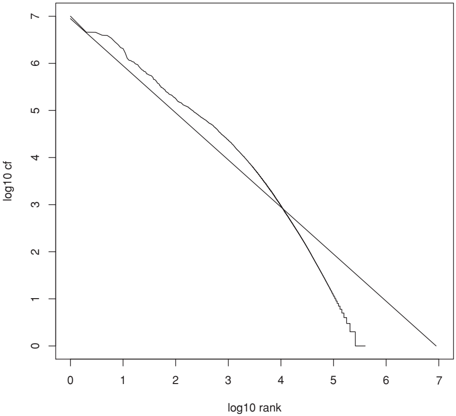
OR → border → lord → morbid → sordid

RD → aboard → ardent → boardroom → border

IIR 04: Index construction



IIR 05: Index compression

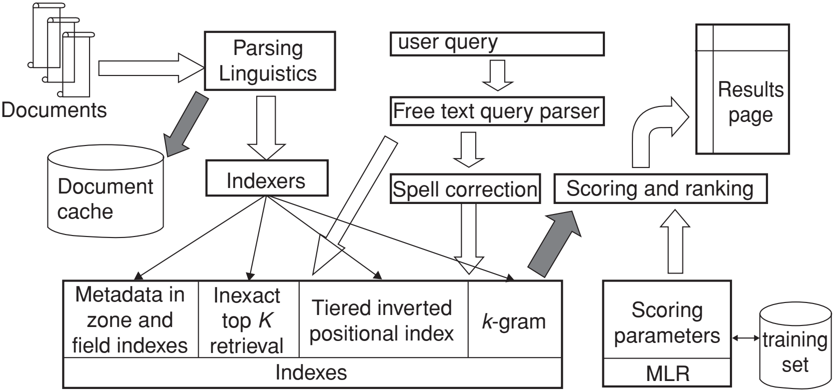


Zipf's law

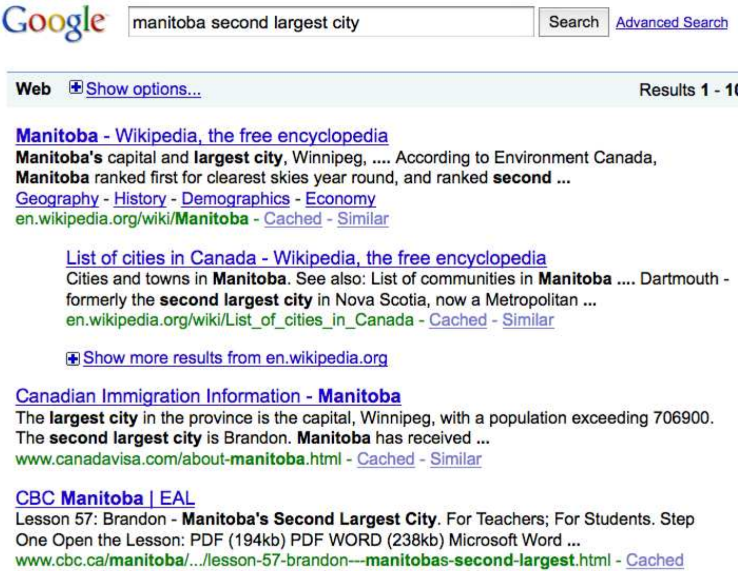
IIR 06: Scoring, term weighting and the vector space model

- Ranking search results
  - Boolean queries only give inclusion or exclusion of documents.
  - For ranked retrieval, we measure the proximity between the query and each document.
  - One formalism for doing this: the vector space model
- Key challenge in ranked retrieval: evidence accumulation for a term in a document
  - 1 vs. 0 occurence of a query term in the document
  - 3 vs. 2 occurences of a query term in the document
  - Usually: more is better
  - But by how much?
  - Need a scoring function that translates frequency into score or weight

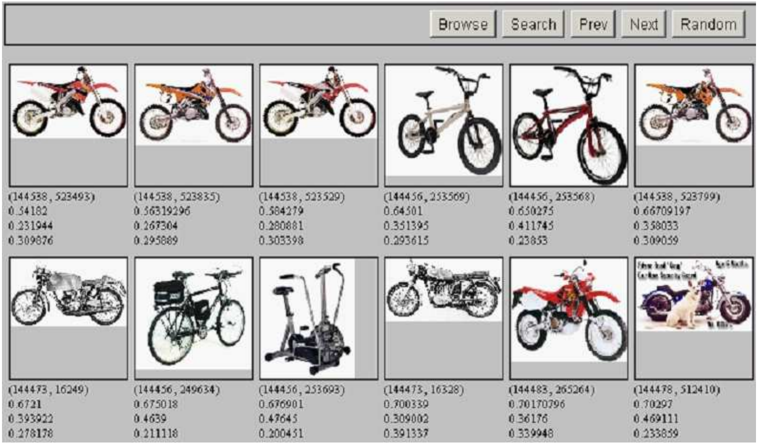
IIR 07: Scoring in a complete search system



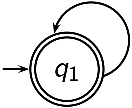
IIR 08: Evaluation and dynamic summaries



IIR 09: Relevance feedback & query expansion



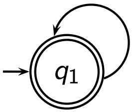
IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

IIR 12: Language models

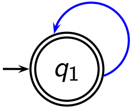


w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

IIR 12: Language models



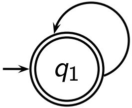
w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

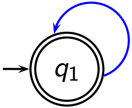
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog

$P(\text{string}) = 0.01$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

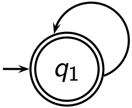
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said

$P(\text{string}) = 0.01$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

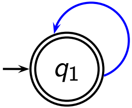
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said

$P(\text{string}) = 0.01 \cdot 0.03$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

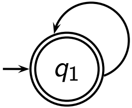
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that

$P(\text{string}) = 0.01 \cdot 0.03$

IIR 12: Language models



w	$P(w \mid q_1)$	w	$P(w \mid q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

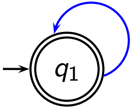
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

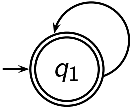
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

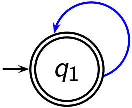
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

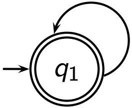
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

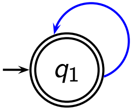
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

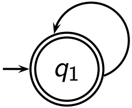
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

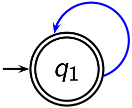
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

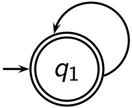
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog STOP

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

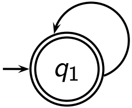
This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog STOP

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$

IIR 12: Language models



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog STOP

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000048$

IIR 13: Text classification & Naive Bayes

- Text classification = assigning documents automatically to predefined classes
- Examples:
  - Language (English vs. French)
  - Adult content
  - Region

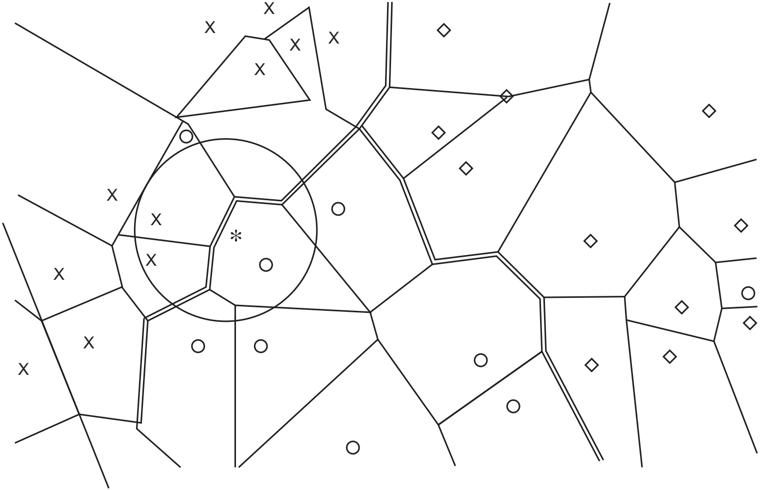
IIR 11: Probabilistic information retrieval

IIR 11: Probabilistic information retrieval

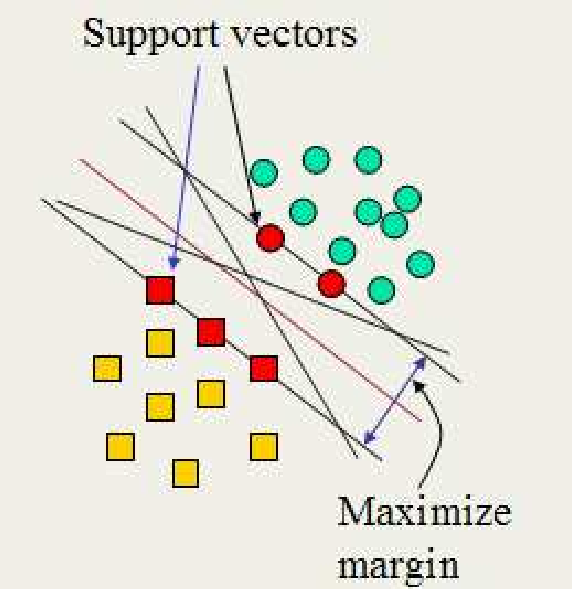
	document	relevant ($R = 1$)	nonrelevant ($R = 0$)
Term present	$x_t = 1$	p_t	u_t
Term absent	$x_t = 0$	$1 - p_t$	$1 - u_t$

$$O(R|\vec{q},\vec{x}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t} \qquad (1)$$

IIR 14: Vector classification



IIR 15: Support vector machines

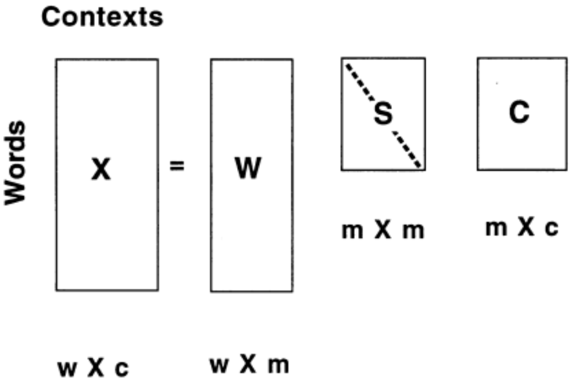


IIR 16: Flat clustering

IIR 17: Hierarchical clustering

http://news.google.com

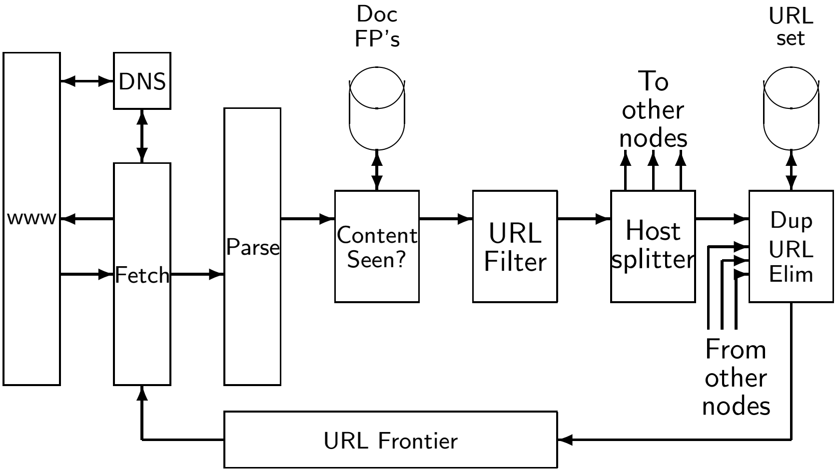
IIR 18: Latent Semantic Indexing



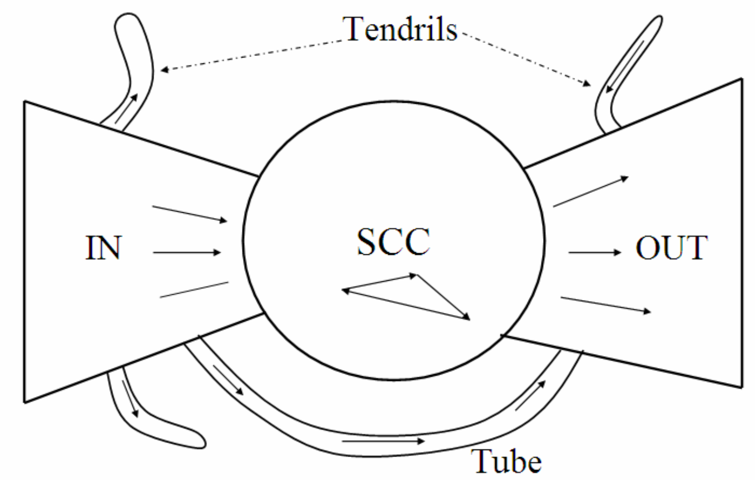
IIR 19: The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users and information needs
- Beyond terms and text: exploit link analysis, user data
- How do web search engines work?
- How can we make them better?

IIR 20: Crawling



IIR 21: Link analysis / PageRank



Take-away

- Administrativa
- Boolean Retrieval: Design and data structures of a simple information retrieval system
- What topics will be covered in this class?

Resources

- Chapter 1 of IIR
- http://ifnlp.org/ir
  - course schedule
  - administrativa
  - information retrieval links
  - Shakespeare search engine