Information Retrieval

Part 1: Introduction
• Some information about your lecturer

• Name
  • Frans van der Sluis

• Background
  • PhD in Interactive Information Retrieval
  • Dual background in Computer Science and Psychology

• Affiliations
  • ATM (project @ UU)
  • Grasp Content (company)
• Some information about your lecture

• Learning goals
  • Define IR in (historic / contemporary) context
  • Understand a standard IR system

• Materials
  • Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008
  • Freely available online at https://nlp.stanford.edu/IR-book/
  • Slides are adapted from Manning's course in Stanford
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
1: VAST QUANTITY OF INFORMATION

• Some facts from… 2012
  • One week of New York Times > a lifetime in 18th century
  • 50mln tweets per day
  • 4 exabytes of unique information generated per year (4 x $10^{19}$ bytes)
  • That’s more than the preceding 5000 years!
2: SPEED AND EASE OF ACCESS

• Some facts from Google
  • 40k queries per second
  • 3.5 billion queries per day
  • 1.2 trillion queries per year
  • A 400ms delay in search results leads to a 0.44% drop in search volume
3: PEOPLE HAVE LOTS OF INFORMATION NEEDS

- Navigational: To go to a particular webpage
  “klm”; “ns”; “nu”; …
- Informational: To learn about certain topics
  “cars”; “information retrieval”; …
- Transactional: To do/find/buy something
  “shop clothes”; “find bar”; “buy IR book”

<table>
<thead>
<tr>
<th>Type of query</th>
<th>Log analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigational</td>
<td>20%</td>
</tr>
<tr>
<td>Informational</td>
<td>48%</td>
</tr>
<tr>
<td>Transactional</td>
<td>30%</td>
</tr>
</tbody>
</table>
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
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).

Q: (Un / semi-) structure data?
Q: Examples of IR applications?
The classic search model

User, task

Info, need

Query

Query refinement

Get rid of mice in a politically correct way

Misconception?

Info about removing mice without killing them

Misformulation?

how trap mice alive

Search engine

Results

Collection
1. **Collection**: A set of documents. Assume it is a static collection for the moment.

2. **Goal**: Retrieve documents with information that is *relevant* to the user's information need and helps the user complete a task.
EVALUATION

• How good are the retrieved docs?
  • **Precision**: Fraction of retrieved docs that are relevant to the user’s information need
  • **Recall**: Fraction of relevant docs in collection that are retrieved

• More precise definitions and measurements to follow later
1. Introduction
   a. Why we need IR
   b. Search context

2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises

3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments

4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments

TABLE OF CONTENTS
IR COMPONENTS

Query

Representation Function

Query Representation

Comparison Function

Results

Documents

Representation Function

Document Representation

Index
Q: Index?
Q: Query?
Q: Comparison?
> grep Brutus *.txt | grep Caesar
julius_caesar.txt:    Except immortal Caesar, speaking of Brutus
julius_caesar.txt:    Brutus and Caesar: what should be in that "Caesar"?
julius_caesar.txt:    "Brutus" will start a spirit as soon as "Caesar."
Caesar doth bear me hard, but he loves Brutus.

ARTEMIDORUS. "Caesar, beware of Brutus; take heed of Cassius;
Flourish. Enter Caesar, Brutus, Cassius, Casca, Decius, Maccetellus, and Marcus Brutus stab Caesar.

him I say that Brutus' love to Caesar was no less than his.
then that friend demand why Brutus rose against Caesar.
So let it be with Caesar. The noble Brutus
For Brutus, as you know

• Using (Linux) GREP
• But what about…
  • Large corpora?
  • Negation? (eg “NOT Calpurnia”)
  • Nearby words? (eg “Julius Caesar”)
  • Different lines?
  • Ranking?

A SUPER-SIMPLE IR SYSTEM
# TABLE OF CONTENTS

1. Introduction  
   a. Why we need IR  
   b. Search context  
2. Basic IR system  
   a. Term-doc index  
   b. Inverted index  
   c. Boolean retrieval model  
   d. Exercises  
3. Index(ing)  
   a. Term vocabulary  
   b. Index creation  
   c. Exercises & assignments  
4. Ranked retrieval  
   a. Take 1: Jaccard scoring  
   b. Take 2: Tf-idf scoring  
   c. Take 3: Vector-space scoring  
   d. Exercises & assignments
## TERM-DOCUMENT INCIDENCE MATRICES

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Query: Brutus AND Caesar NOT Calpurnia

1 if play contains word, 0 otherwise
So we have a 0/1 vector for each term

To answer:
- Take 0/1 vectors for Brutus, Caesar, Calpurnia (complemented)
- Bitwise AND

The answer:
- 110100 AND
- 110111 AND
- 101111 =
- 100100

<table>
<thead>
<tr>
<th></th>
<th>Antony</th>
<th>Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
• Antony 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.

ANSWERS TO QUERY
BIGGER COLLECTIONS

• Vast quantities of information?

• Nope: Huge matrix
  • Consider 500k distinct words, 1mln documents
  • 500k x 1mln matrix has half-a-trillion 0’s and 1’s
• And: Very sparse
  • Consider 1k words per doc
  • Only 1bln 1’s! Versus 499bln 0’s.
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
The key data structure underlying modern IR
- We only record where the 1’s are
- In variable-size postings lists per term
- Identify each doc by a docID, a document serial number

INVERTED INDEX
• For each term $t$, store a list of all documents that contain $t$.
• Q: Why is this called an “inverted” index?

```
Brutus  ⟩→ 1 2 4 11 31 45 173 174
Caesar  ⟩→ 1 2 4 5 6 16 57 132 ...
Calpurnia⟩→ 2 31 54 101
:
```

dictionary file

postings file
SIMPLE CONJUNCTIVE QUERY

• Consider the query: BRUTUS AND CALPURNIA
• To find all matching documents using inverted index:
  • Locate BRUTUS in the dictionary
  • Retrieve its postings list from the postings file
  • Locate CALPURNIA in the dictionary
  • Retrieve its postings list from the postings file
  • Intersect the two postings lists
  • Return intersection to user
INTERSECTING TWO POSTING LISTS

• This is time linear $O(x+y)$ in the length of the postings lists.
• Note: This only works if postings lists are sorted.

Brutus: $\rightarrow$ 1→2→4→11→31→45→173→174
Calpurnia: $\rightarrow$ 2→31→54→101
Intersection: $\Rightarrow$ 2→31
INTERSECTING TWO POSTING LISTS

\[
\text{INTERSECT}(p_1, p_2)
\]

1. \(answer \leftarrow \langle \rangle\)
2. \(\textbf{while } p_1 \neq \text{NIL} \text{ and } p_2 \neq \text{NIL}\)
3. \(\textbf{do if } \text{docID}(p_1) = \text{docID}(p_2)\)
4. \(\quad \textbf{then } \text{ADD}(\text{answer}, \text{docID}(p_1))\)
5. \(\quad p_1 \leftarrow \text{next}(p_1)\)
6. \(\quad p_2 \leftarrow \text{next}(p_2)\)
7. \(\textbf{else if } \text{docID}(p_1) < \text{docID}(p_2)\)
8. \(\quad \textbf{then } p_1 \leftarrow \text{next}(p_1)\)
9. \(\quad \textbf{else } p_2 \leftarrow \text{next}(p_2)\)
10. \(\textbf{return } answer\)
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. **Boolean retrieval model**
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
Perhaps the simplest model to build an IR system on

Queries are Boolean expressions:
  • Using AND, OR and NOT to join query terms
  • Views each document as a set of words
  • Is precise: document matches condition or not
  • But: all or nothing / no ranking!

Primary commercial retrieval model for 3 decades
Still lots in use nowadays (eg email, library catalogs, OS X Spotlight)
What is the best order for query processing?

Process in order of increasing freq:
- Start with smallest set, then keep cutting further.
- AND operator cuts set size
- Note: Needs doc freq in dictionary!

Estimate the size of each OR by the sum of its doc. freq.’s (conservative).
EXAMPLE: WESTLAW

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992; new federated search added 2010)
- Tens of terabytes of data; ~700,000 users
- Majority of users *still* use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
    - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
- /3 = within 3 words, /S = in same sentence
- Note that SPACE is disjunction (OR), not conjunction (AND)!
Many professional searchers still like Boolean search
- You know exactly what you are getting
- Long, precise queries; proximity operators; incrementally developed; not like web search

When are Boolean queries the best way of searching?
- Depends on: information need, searcher, document collection, . . .
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments

TABLE OF CONTENTS
EXERCISE: BOOLEAN RETRIEVAL

Exercise: Compute hit list for

\(((\text{paris} \text{ AND NOT france}) \text{ OR lear})\)
EXERCISE: MORE BOOLEAN QUERIES

• Exercise: Adapt the merge strategy for the queries
  Brutus AND NOT Caesar
  Brutus OR NOT Caesar

• Can we still run through the merge in time $O(x+y)$? What can we achieve?
EXERCISE: QUERY OPTIMISATION

• Recommend a query processing order for:
  (tangerine OR trees) AND
  (marmalade OR skies) AND
  (kaleidoscope OR eyes)

• Which two terms should we process first?

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>eyes</td>
<td>213312</td>
</tr>
<tr>
<td>kaleidoscope</td>
<td>87009</td>
</tr>
<tr>
<td>marmalade</td>
<td>107913</td>
</tr>
<tr>
<td>skies</td>
<td>271658</td>
</tr>
<tr>
<td>tangerine</td>
<td>46653</td>
</tr>
<tr>
<td>trees</td>
<td>316812</td>
</tr>
</tbody>
</table>
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
INDEX CONSTRUCTION

Documents to be indexed

Tokenizer

Token stream

Linguistic modules

Modified tokens

Indexer

Inverted index

Friends, Romans, countrymen.

friend

roman

countryman

friend

roman

countryman

\[ 2 \rightarrow 4 \]

\[ 1 \rightarrow 2 \]

\[ 13 \rightarrow 16 \]
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
TERM REPRESENTATION

- **Tokenization**
  - Cut character sequence into word tokens
  - Deal with “John’s”, a state-of-the-art solution

- **Normalization**
  - Map text and query term to same form
  - You want U.S.A. and USA to match

- **Stemming**
  - We may wish different forms of a root to match
  - authorize, authorization

- **Stop words**
  - We may omit very common words (or not)
  - the, a, to, of
Input: “Friends, Romans, and Countrymen”
Output: Tokens
  • Friends
  • Romans
  • Countrymen

A token is an instance of a sequence of characters
But, what are valid tokens to emit?
TOKENISATION CHALLENGES

• Finland’s capital ->
  Finland + s ? Finlands? Finland's?
• Hewlett-Packard ->
  Hewlett + Packard?
• State-of-the-art ->
  • State + of + the + art?
• The Hague ->
  • The + Hague? The Hague?

• Q: Anybody?
TOKENISATION CHALLENGES

• Numbers
  • 3/20/91    Mar. 12, 1991    20/3/91
  • 55 B.C.
  • Key is 324a3df234cb23e
  • (800) 234-2333
  • Often ignored in older IR systems, even though valuable!

• Noun compounds
  • Lebensversicherungsgesellschaftsangestellter
  • ‘life insurance company employee’
  • German retrieval systems benefit greatly from a compound splitter module
  • Can give a 15% performance boost for German
STOP WORDS REMOVAL

• With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
  • They have little semantic content: the, a, and, to, be
  • There are a lot of them: ~30% of postings for top 30 words
• But the trend is away from doing this:
  • Good compression techniques
  • Good query optimisation techniques
• You need them for:
  • Phrase queries: “King of Denmark”
  • Various song titles, etc.: “Let it be”, “To be or not to be”
  • “Relational” queries: “flights to London”
NORMALISATION

• Results in (index dictionary) terms
• Need to normalise identically for query and index!!
• We most commonly implicitly define equivalence classes, eg:

  • Deleting periods to form a term:
    U.S.A., USA -> USA
  • Deleting hyphens to form a term:
    anti-discriminatory, antidiscriminatory -> antidiscriminatory
NORMALISATION

• Accents:
  résumé -> resume

• Umlauts:
  Tuebingen, Tübingen, Tubingen -> Tubingen

• Lower case:
  General Motors -> general motors
  Fed -> fed
  SAIL -> sail
NORMALISATION CHALLENGES

• But… Longstanding Google example: [fixed in 2011…]
  • Query C.A.T.
  • #1 result is for “cats” (well, Lolcats) not Caterpillar Inc.

• Alternative to equivalence classes is to do “asymmetric query expansion”.
  Eg.
  • Enter: window Search: window, windows
  • Enter: windows Search: Windows, windows, window
  • Enter: Windows Search: Windows
• Potentially more powerful, but less efficient
• Reduce terms to their “roots” before indexing
  • “Stemming” suggests crude affix chopping
  • Language dependent
  • E.g., automate(s), automatic, automation -> automat.

For example compressed and compression are both accepted as equivalent to compress.

For example compress and compress are both accepted as equivalent to compress.
• Eg Porter’s stemming algorithm
• Typical rules
  - sses → ss
  - ies → i
  - ational → ate
  - tional → tion

• Weight-of-word sensitive rules
  - \((m>1)\) EMENT → 
• Examples
  - replacement → replac
  - cement → cement
DOES STEMMING HELP?

• English: very mixed results. Helps recall for some queries but harms precision on others
  • E.g., operative (dentistry) ⇒ oper

• Definitely useful for Spanish, German, Finnish, …
  • 30% performance gains for Finnish!
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
Sequence of (Modified token, Document ID) pairs.

Doc 1
I did enact Julius Caesar I was killed i' the Capitol;

Doc 2
So let it be with Caesar. The noble Brutus hath told you Caesar was
• Sort by terms
• And then by docID

• Core indexing steps!
CREATE LISTS, ADD COUNTS

• Multiple term entries in a single document are merged
• Split into dict and postings
• Doc. freq. is added
WHERE DO WE PAY IN STORAGE

- IR system implementation
  - How do we index efficiently?
  - How much storage do we need?

**Terms and counts**

- ambitious: 1
- be: 1
- brutus: 2
- capitol: 1
- caesar: 2
- did: 1
- enact: 1
- hath: 1
- i: 1
- ’i: 1
- it: 1
- julius: 1
- killed: 1
- let: 1
- me: 1
- noble: 1
- so: 1
- the: 2
- told: 1
- you: 1
- was: 2
- with: 1

**Pointers**

**Lists of docIDs**

- term: doc. freq. → postings lists
  - ambitious: 1 → 2
  - be: 1 → 2
  - brutus: 2 → 1
  - capitol: 1 → 1
  - caesar: 2 → 1
  - did: 1 → 1
  - enact: 1 → 1
  - hath: 1 → 1
  - i: 1 → 1
  - ’i: 1 → 1
  - it: 1 → 1
  - julius: 1 → 1
  - killed: 1 → 1
  - let: 1 → 1
  - me: 1 → 1
  - noble: 1 → 1
  - so: 1 → 1
  - the: 2 → 1
  - told: 1 → 1
  - you: 1 → 1
  - was: 2 → 1
  - with: 1 → 1
SPLIT INTO DICT AND POSTINGS FILES

Brutus → 1 2 4 11 31 45 173 174

Caesar → 1 2 4 5 6 16 57 132 ...

Calpurnia → 2 31 54 101

::

<table>
<thead>
<tr>
<th>dictionary</th>
<th>postings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments

TABLE OF CONTENTS
1. Why is it so important to sort the dict and postings?
2. Write down three queries you did recently:
   a. Was it navigational, informational, or transactional?
   b. How was it affected by (and/or):
      a. Stop words removal
      b. Stemming
      c. Lower case normalisation
      d. Dots/hyphens normalisation
SELECTED ASSIGNMENTS

• Ch 1. exercises
  • Postings list and incidence matrix
    • 1.2, 1.3
  • Boolean retrieval
    • 1.9, 1.10, 1.11
  • Stemming and normalisation
    • 2.1, 2.3
TABLE OF CONTENTS

1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises & assignments
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises & assignments
Thus far, our queries have all been Boolean.

- Documents either match or don’t.
- Good for expert users
  - With precise understanding of their needs and the collection.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don’t want to wade through 1000s of results.
    - This is particularly true of web search.
Boolean queries often result in either too few (=0) or too many (1000s) results.

Query 1: “standard user dlink 650” → 200,000 hits
Query 2: “standard user dlink 650 no card found”: 0 hits

It takes a lot of skill to come up with a query that produces a manageable number of hits.

• AND gives too few; OR gives too many
RANKED RETRIEVAL MODELS

• Ranked retrieval
  • From a set of documents satisfying a query expression
  • To an ordering over the (top) documents given a query

• Free text queries
  • From a query language of operators and expressions
  • To one or more words in a human language

• Q: Do these always go together?
FEAST OR FAMINE: NOT A PROBLEM IN RANKED RETRIEVAL

• When a system produces a ranked result set, large result sets are not an issue
  • Indeed, the size of the result set is not an issue
  • We just show the top $k$ (≈ 10) results
  • We don’t overwhelm the user

• Premise: the ranking algorithm works
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsections</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>a. Why we need IR</td>
</tr>
<tr>
<td></td>
<td>b. Search context</td>
</tr>
<tr>
<td>2. Basic IR system</td>
<td>a. Term-doc index</td>
</tr>
<tr>
<td></td>
<td>b. Inverted index</td>
</tr>
<tr>
<td></td>
<td>c. Boolean retrieval model</td>
</tr>
<tr>
<td></td>
<td>d. Exercises</td>
</tr>
<tr>
<td>3. Index(ing)</td>
<td>a. Term vocabulary</td>
</tr>
<tr>
<td>4. Ranked retrieval</td>
<td>b. Index creation</td>
</tr>
<tr>
<td></td>
<td>c. Exercises</td>
</tr>
<tr>
<td></td>
<td><strong>a. Take 1: Jaccard scoring</strong></td>
</tr>
<tr>
<td></td>
<td>b. Take 2: Tf-idf scoring</td>
</tr>
<tr>
<td></td>
<td>c. Take 3: Vector-space scoring</td>
</tr>
<tr>
<td></td>
<td>d. Exercises</td>
</tr>
</tbody>
</table>
We wish to return in order the documents most likely to be useful to the searcher.

How can we rank-order the documents in the collection with respect to a query?

Assign a score – say in $[0, 1]$ – to each document.

This score measures how well document and query "match".

Take 1: Jaccard scoring.
• Jaccard: A commonly used measure of overlap of two sets A and B (i.e., sets of terms)

\[ jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|} \]

\[ jaccard(A,A) = 1 \]

\[ jaccard(A,B) = 0 \text{ if } A \cap B = 0 \]

• A and B don’t have to be the same size.

• Always assigns a number between 0 and 1.

**TAKE 1: JACCARD COEFFICIENT**
Exercise: What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?

Query: *ides of march*

**Document 1:** *caesar died in march*

**Document 2:** *the long march*
ISSUES WITH JACCARD FOR SCORING

• Jaccard scoring ignores
  • How many times a term occurs in a document (*term frequency*)
  • That rare terms in a collection are more informative than frequent terms (*document frequency*)

• We need a more sophisticated way of normalizing for length
• Later in this lecture, we’ll use $|A \cap B|/\sqrt{|A \cup B|}$ instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalisation.
<table>
<thead>
<tr>
<th>1. Introduction</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Why we need IR</td>
<td>b. Search context</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Basic IR system</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Term-doc index</td>
<td>b. Inverted index</td>
<td>c. Boolean retrieval model</td>
<td>d. Exercises</td>
</tr>
<tr>
<td>3. Index(ing)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. Term vocabulary</td>
<td>b. Index creation</td>
<td>c. Exercises</td>
<td></td>
</tr>
<tr>
<td>4. Ranked retrieval</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RECALL: TERM-DOC INCIDENCE MATRIX

<table>
<thead>
<tr>
<th>Document</th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Each document is represented by a binary vector $\mathbf{v} \in \{0, 1\}^{|V|}$
TERM-DOCUMENT COUNT MATRICES

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in \( \mathbb{N}^v \): a column below

<table>
<thead>
<tr>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Brutus</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>worser</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Vector representation doesn’t consider the ordering of words in a document. John is quicker than Mary and Mary is quicker than John have the same vectors.

This is called the bag of words model.

Later: We could extend the index with positional information (a positional index) as well.
Take 2a: Use tf when computing query-document match scores

The term frequency $tf_{t,d}$ of term $t$ in document $d$ is defined as the number of times that $t$ occurs in $d$.

But, raw term frequency is not what we want:
- A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
- But not 10 times more relevant.

NB: frequency = count in IR
LOG-FREQUENCY WEIGHTING

• The log frequency weight of term $t$ in $d$ is

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

• $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.

• Score for a document-query pair:
  • Score = sum over terms $t$ in both $q$ and $d$: $= \sum_{t \in q \cap d} (1 + \log tf_{t,d})$

• The score is 0 if none of the query terms is present in the document.
Take 2b: Use DF when computing query-document match scores.

$df_t$ is the document frequency of $t$: the number of documents that contain $t$.

Rationale

- Rare terms are more informative than frequent terms.
- Frequent terms are less informative than rare terms (cf. stop words).
Example

A very rare term: arachnocentric

Vs. frequent terms: high, increase, line

Which term has more discriminative power?

A document containing a rare term is more likely to be relevant than a document that doesn’t

But it’s not a sure indicator of relevance.

We will use document frequency (df) to capture this.
• $df_t$ is the document frequency of $t$: the number of documents that contain $t$
  
  $df_t$ is an inverse measure of the informativeness of $t$
  $df_t \leq N$

• We define the idf (inverse document frequency) of $t$ by

$$idf_t = \log_{10} \left( \frac{N}{df_t} \right)$$

• We use log $(N/df_t)$ instead of $N/df_t$ to “dampen” the effect of idf.

• The exact log base is immaterial
IDF EXAMPLE

SUPPOSE $N = 1$ MILLION

<table>
<thead>
<tr>
<th>term</th>
<th>$df_t$</th>
<th>$idf_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>calpurnia</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>animal</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>sunday</td>
<td>1,000</td>
<td></td>
</tr>
<tr>
<td>fly</td>
<td>10,000</td>
<td></td>
</tr>
<tr>
<td>under</td>
<td>100,000</td>
<td></td>
</tr>
<tr>
<td>the</td>
<td>1,000,000</td>
<td></td>
</tr>
</tbody>
</table>

$$idf_t = \log_{10} \left( \frac{N}{df_t} \right)$$

There is one idf value per term $t$ in a collection.
IDF EXAMPLE

SUPPOSE $N = 1$ MILLION

<table>
<thead>
<tr>
<th>term</th>
<th>$df_t$</th>
<th>$idf_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>calpurnia</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>animal</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>sunday</td>
<td>1,000</td>
<td>3</td>
</tr>
<tr>
<td>fly</td>
<td>10,000</td>
<td>2</td>
</tr>
<tr>
<td>under</td>
<td>100,000</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1,000,000</td>
<td>0</td>
</tr>
</tbody>
</table>

$idf_t = \log_{10} \left( \frac{N}{df_t} \right)$

There is one idf value per term $t$ in a collection.
EFFECT OF IDF ON RANKING

• Q: Does idf have an effect on ranking for one-term queries, like iPhone?
• A: No
  • idf affects the ranking of documents for queries with at least two terms
  • For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.
Why not collection frequency?

The collection frequency of \( t \) is the number of occurrences of \( t \) in the collection, counting multiple occurrences.

Example query: “try to buy insurance”

<table>
<thead>
<tr>
<th>Word</th>
<th>Collection frequency</th>
<th>Document frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>insurance</td>
<td>10440</td>
<td>3997</td>
</tr>
<tr>
<td>try</td>
<td>10422</td>
<td>8760</td>
</tr>
</tbody>
</table>
TAKE 2: TF-IDF WEIGHTING

• Take 2: The tf-idf weight of a term is the product of its tf weight and its idf weight.

\[ w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / df_t) \]

• Best known weighting scheme in information retrieval
  • Increases with the number of occurrences within a document
  • Increases with the rarity of the term in the collection
• Note: “-“ in name is a hyphen!
SCORE FOR A DOCUMENT GIVEN A QUERY

\[
\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf}.\text{idf}_{t,d}
\]

- There are many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - ...

Sec. 6.2.2
# TABLE OF CONTENTS

1. Introduction  
   a. Why we need IR  
   b. Search context  
2. Basic IR system  
   a. Term-doc index  
   b. Inverted index  
   c. Boolean retrieval model  
   d. Exercises  
3. Index(ing)  
   a. Term vocabulary  
4. Ranked retrieval  
   a. Take 1: Jaccard scoring  
   b. Take 2: Tf-idf scoring  
   c. **Take 3: Vector-space scoring**  
   d. Exercises
### BINARY $\rightarrow$ COUNT $\rightarrow$ WEIGHT MATRIX

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>5.25</td>
<td>3.18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.35</td>
</tr>
<tr>
<td>Brutus</td>
<td>1.21</td>
<td>6.1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>8.59</td>
<td>2.54</td>
<td>0</td>
<td>1.51</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1.54</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>2.85</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1.51</td>
<td>0</td>
<td>1.9</td>
<td>0.12</td>
<td>5.25</td>
<td>0.88</td>
</tr>
<tr>
<td>worser</td>
<td>1.37</td>
<td>0</td>
<td>0.11</td>
<td>4.15</td>
<td>0.25</td>
<td>1.95</td>
</tr>
</tbody>
</table>

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$
So we have a $|V|$-dimensional vector space

- Terms are axes of the space
- Documents are points or vectors in this space

- Very high-dimensional: tens of millions of dimensions
- Very sparse vectors - most entries are zero.
**QUERIES AS VECTOR**

- Vector space retrieval model
  - **Key idea 1:** Queries as vectors in the space
  - **Key idea 2:** Rank documents according to their proximity to the query in this space

- Proximity
  - $\text{proximity} = \text{similarity of vectors}$
  - $\text{proximity} \approx \text{inverse of distance}$
The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.
USE ANGLE INSTEAD OF DISTANCE

• Thought experiment: take a document \( d \) and append it to itself. Call this document \( d' \).
  “Semantically” \( d \) and \( d' \) have the same content

• The Euclidean distance between the two documents can be quite large
• The angle between the two documents is 0, corresponding to maximal similarity.

• Key idea: Rank documents according to angle with query.
The following two notions are equivalent.

- Rank documents in \textit{decreasing} order of the angle between query and document
- Rank documents in \textit{increasing} order of \( \cos(\text{query, document}) \)

Cosine is a monotonically decreasing function for the interval \([0^\circ, 180^\circ]\)
• But how – *and why* – should we be computing cosines?
• L_2 norm gives length normalisation:

\[ \| \vec{x} \|_2 = \sqrt{\sum_i x_i^2} \]

• Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)

• Effect on the two documents d and d’ (d appended to itself) from earlier slide?
COSINE(QUERY, DOCUMENT)

- \(q_i\) is the tf-idf weight of term \(i\) in the query
- \(d_i\) is the tf-idf weight of term \(i\) in the document

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

- \(\cos(\vec{q}, \vec{d})\) is the cosine similarity of \(\vec{q}\) and \(\vec{d}\) … or,
- equivalently, the cosine of the angle between \(\vec{q}\) and \(\vec{d}\).
For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

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

for \( q, d \) length-normalized.
• Docs and query as vectors
• Unit (L2) length normalisation
• Rank by inverse angle
  • Lower angle $\approx$ higher cosine

**COSINE SIMILARITY ILLUSTRATED**
WEIGHTING MAY DIFFER IN QUERIES VS DOCUMENTS

• Weighting can differ for docs and queries
  • SMART Notation: **ddd.qqq**
  • Many variants possible

• A very standard weighting scheme is: **Inc.ltc**
  • Doc:
    (L) logarithmic term freq - (N) no idf - (C) cosine normalisation (L2)
  • Query:
    (L) logarithmic term freq - (T) idf - (C) cosine normalisation (L2)

A bad idea?
TF-IDF EXAMPLE: LNC.LTC

Document: *car insurance auto insurance*
Query: *best car insurance*

<table>
<thead>
<tr>
<th>Term</th>
<th>Query</th>
<th>Document</th>
<th>Prod</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wt</td>
<td>df</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>best</td>
<td>1</td>
<td>1</td>
<td>50000</td>
</tr>
<tr>
<td>car</td>
<td>1</td>
<td>1</td>
<td>10000</td>
</tr>
<tr>
<td>insurance</td>
<td>1</td>
<td>1</td>
<td>1000</td>
</tr>
</tbody>
</table>

Score = 0 + 0 + 0.27 + 0.53 = 0.8

Doc length = \(\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92\)
Computing Cosine Scores

CosineScore(q)
1. float Scores[N] = 0
2. float Length[N]
3. for each query term t
4. do calculate $w_{t,q}$ and fetch postings list for $t$
5. for each pair $(d, tf_{t,d})$ in postings list
6. do $Scores[d] += w_{t,d} \times w_{t,q}$
7. Read the array Length
8. for each $d$
9. do $Scores[d] = Scores[d]/Length[d]$
10. return Top K components of Scores[]
1. Introduction
   a. Why we need IR
   b. Search context
2. Basic IR system
   a. Term-doc index
   b. Inverted index
   c. Boolean retrieval model
   d. Exercises
3. Index(ing)
   a. Term vocabulary
   b. Index creation
   c. Exercises
4. Ranked retrieval
   a. Take 1: Jaccard scoring
   b. Take 2: Tf-idf scoring
   c. Take 3: Vector-space scoring
   d. Exercises
QUESTIONS

1. Do we need to do length normalisation for the query?
2. What happens when we apply idf weighting to both query and doc vectors?
   Hint: See the cosine similarity calculation for length-normalized vectors
3. And why would we want that?
How similar are the novels **SaS**: *Sense and Sensibility*, **PaP**: *Pride and Prejudice*, and **WH**: *Wuthering Heights*?

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>115</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>jealous</td>
<td>10</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>gossip</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>38</td>
</tr>
</tbody>
</table>

Term frequencies (counts)

Note: To simplify this exercise, we don’t do idf weighting.
• Exercise:
  1. Log frequency weighting
  2. Length normalisation
  3. $\cos(\text{SaS}, \text{PaP})$ and $\cos(\text{SaS}, \text{WH})$

**EXERCISE: COSINE SIMILARITY**

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>115</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>jealous</td>
<td>10</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>gossip</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>38</td>
</tr>
</tbody>
</table>

Term frequencies (counts)

$$w_{t,d} = \begin{cases} 
1 + \log_{10} tf_{t,d}, & \text{if } tf_{t,d} > 0 \\
0, & \text{otherwise}
\end{cases}$$
## Exercise: Cosine Similarity

- **Exercise:**
  1. Log frequency weighting
  2. Length normalisation
  3. \( \cos(SaS, PaP) \) and \( \cos(SaS, WH) \)

### Log frequency weights

\[
\| \bar{x} \|_2 = \sqrt{\sum x_i^2}
\]

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>3.06</td>
<td>2.76</td>
<td>2.30</td>
</tr>
<tr>
<td>jealous</td>
<td>2.00</td>
<td>1.85</td>
<td>2.04</td>
</tr>
<tr>
<td>gossip</td>
<td>1.30</td>
<td>0</td>
<td>1.78</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>2.58</td>
</tr>
</tbody>
</table>
### Exercise: Cosine Similarity

#### Exercise:
1. Log frequency weighting
2. Length normalisation
3. \( \cos(SaS, \text{PaP}) \) and \( \cos(SaS, \text{WH}) \)

#### Log frequency weighting

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>0.789</td>
<td>0.832</td>
<td>0.524</td>
</tr>
<tr>
<td>jealous</td>
<td>0.515</td>
<td>0.555</td>
<td>0.465</td>
</tr>
<tr>
<td>gossip</td>
<td>0.335</td>
<td>0.0</td>
<td>0.405</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>0.588</td>
</tr>
</tbody>
</table>

#### Length normalisation

Unit vectors

\[
\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_{i=1}^{\|V\|} q_i d_i
\]
### Exercise: Cosine Similarity

**Exercise:**

1. Log frequency weighting
2. Length normalisation
3. \( \cos(SaS, PaP) \) and \( \cos(SaS, WH) \)

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>0.789</td>
<td>0.832</td>
<td>0.524</td>
</tr>
<tr>
<td>jealous</td>
<td>0.515</td>
<td>0.555</td>
<td>0.465</td>
</tr>
<tr>
<td>gossip</td>
<td>0.335</td>
<td>0.0</td>
<td>0.405</td>
</tr>
<tr>
<td>wuthering</td>
<td>0.0</td>
<td>0.0</td>
<td>0.588</td>
</tr>
</tbody>
</table>

\[
\cos(SaS, PaP) \approx 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94
\]

\[
\cos(SaS, WH) \approx 0.79
\]

\[
\cos(PaP, WH) \approx 0.69
\]
ASSIGNMENTS

• Ch6 exercises
  • Tf-idf
    Note: These are w/o term weighting
    • 6.8, 6.9, 6.10, 6.11, 6.12
    • 6.15, 6.16, 6.17
  • Vector space
    • 6.19, 6.20, 6.22
• \( \log(x = 1) = \log(x = N/N) = 0 \) no matter the base
• \( \log(1 < x < 2) \) differs
  higher base -> faster weight increase
• \( \log(x > 2) \) differs surprisingly little, eg
  \[ \log_2: \frac{\log(10)}{\log(2)} = \frac{3.32}{1} = 3.32 \]
  \[ \log_{10}: \frac{\log(10)}{\log(2)} = \frac{1}{0.30} = 3.32 \]
EXERCISE 6.16: PYTHAGORAS THEOREM

\[ 11^2 + 11^2 = R^2 \]
\[ 242 = R^2 \]
\[ 15.6 = R \]
<table>
<thead>
<tr>
<th>1. Introduction</th>
<th>b. Index creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Why we need IR</td>
<td>c. Exercises</td>
</tr>
<tr>
<td>b. Search context</td>
<td></td>
</tr>
<tr>
<td>2. Basic IR system</td>
<td>4. Ranked retrieval</td>
</tr>
<tr>
<td>a. Term-doc index</td>
<td>a. Take 1: Jaccard scoring</td>
</tr>
<tr>
<td>b. Inverted index</td>
<td>b. Take 2: Tf-idf scoring</td>
</tr>
<tr>
<td>c. Boolean retrieval model</td>
<td>c. Take 3: Vector-space scoring</td>
</tr>
<tr>
<td>d. Exercises</td>
<td>d. Exercises</td>
</tr>
<tr>
<td>3. Index(ing)</td>
<td></td>
</tr>
<tr>
<td>a. Term vocabulary</td>
<td></td>
</tr>
</tbody>
</table>