Information Retrieval

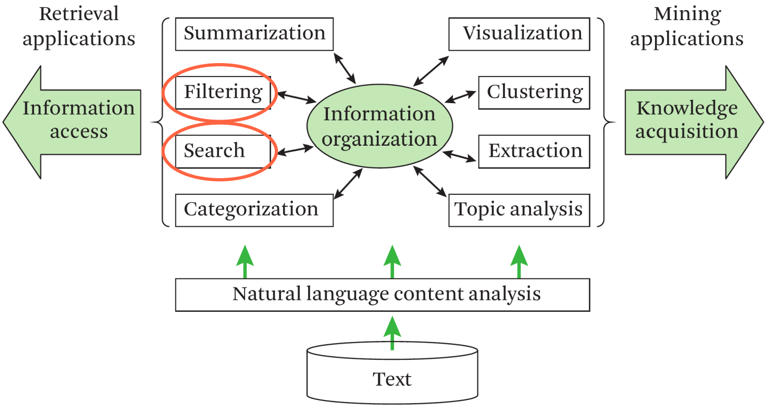
Marcel Bollmann

Department of Computer and Information Science (IDA)

LINKÖPING UNIVERSITY

Adapted from slides by Marco Kuhlmann.

Reminder: Conceptual framework



Zhai and Massung (2016)

What is Information Retrieval?

1

Defining "information retrieval"

Information retrieval (IR) is

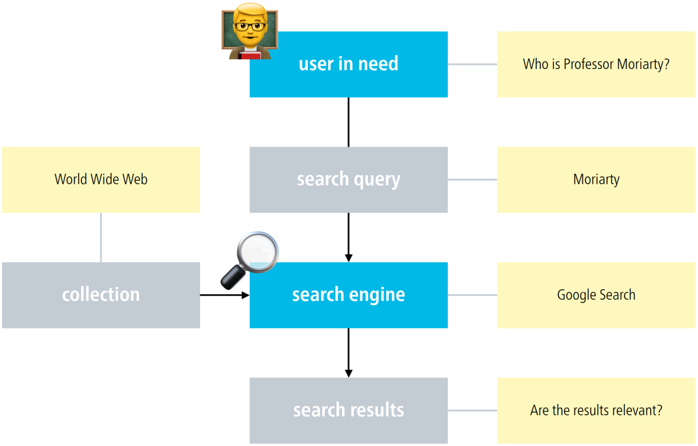
- finding material (usually documents)
- of an unstructured nature (usually text)
- that satisfies an information need
- from within large collections (usually stored on computers).

Manning, Raghavan, and Schütze (2009), p.1

The basic search model

- User formulates a **search query** to communicate their "information need".
- The IR system finds documents in the collection that **match** the search query.
- A *good* IR system also considers **relevance** for the user's information need.
  - Raises the question: How to *evaluate* relevance?

The basic search model



Adapted from Manning, Raghavan, and Schütze (2009)

Boolean retrieval

: *Which Sherlock Holmes stories contain the word "Moriarty"?*

- The **Boolean retrieval model** is perhaps the simplest and historically the most widely used model in IR.
- A query in this model is a normal-form Boolean expression.
  - Atoms correspond to **terms** ("words")
  - An atom t is true for a document d *iff* (if and only if) t occurs in d.

Boolean retrieval: Example

: *Which Sherlock Holmes stories contain the word “Moriarty” and “Lestrade”, but not the word “Adair”?*

↓

As a Boolean query

Moriarty AND Lestrade AND NOT Adair

Term-document matrix

In a **term–document matrix**:

- the rows correspond to **search terms** t
- the columns correspond to **documents** d
- a cell (t, d) is 1 if t occurs in d, and 0 otherwise

Term-document matrix

	Scandal in Bohemia	Final problem	Empty house	Norwood builder	Dancing men	Retired colourman
Adair	0	0	1	0	0	0
Adler	1	0	0	0	0	0
Lestrade	0	0	1	1	0	0
Moriarty	0	1	1	1	0	0

Moriarty AND Lestrade AND NOT Adair

	Scandal in Bohemia	Final problem	Empty house	Norwood builder	Dancing men	Retired colourman
Adair	0	0	1	0	0	0
Adler	1	0	0	0	0	0
Lestrade	0	0	1	1	0	0
Moriarty	0	1	1	1	0	0

Ranked retrieval

- Boolean retrieval is "black or white": a document either matches the query or not.
- A **ranked retrieval** system assigns **scores** to documents based on *how well* they match a given search query.
  - Search results can then be *ranked* based on their score with respect to the query.

Consider...

- A document containing *Moriarty* and *Lestrade* 20 times each
- A document containing *Moriarty* and *Lestrade* only once

Important concepts

- search query
- relevance
- Boolean retrieval
- ranked retrieval

Text Preprocessing

2

Text preprocessing

- Almost any time we work with textual data, we need to think about **how to preprocess** our documents.

- **Tokenization**: splitting up text into smaller units (tokens), e.g. words

- Other **preprocessing steps** typically come after tokenization, for example:
  - filtering punctuation marks
  - stop word removal
  - lowercasing
  - lemmatization

Tokenization

- Based on whitespace:

```python
text = 'Dwarves are "intelligent", alcohol-dependent creatures'
for token in text.split():
    print(token)
```

- Using spaCy:

```python
import spacy
nlp = spacy.load('en_core_web_sm')
for token in nlp(text):
    print(token)
```

Text from the Dwarf Fortress wiki

Tokenization

Before tokenization

Dwarves **(singular, Dwarf)** are **“intelligent”, alcohol-dependent,** humanoid creatures that are the featured race of fortress **mode,** as well as being playable in adventurer **mode.**

After tokenization

Dwarves **(singular , Dwarf)** are **“ intelligent ” , alcohol - dependent ,** humanoid creatures that are the featured race of fortress **mode ,** as well as being playable in adventurer **mode .**

Stop words

- A **stop word** is a frequent word that does not contribute much to a given task.
  - Typical examples: *a, the, and, is, are, …*

```
>>> [(token, token.is_stop) for token in nlp(text)]
[(Dwarves, False), (are, True), (", False), (intelligent, False), ...]
```

- Stop words are **application-specific** – there is no single universal list!
  - Stop word removal may even be disadvantageous.

Stop word removal

Before removal

Dwarves (singular , Dwarf) are " intelligent " , alcohol - dependent , humanoid creatures that are the featured race of fortress mode , as well as being playable in adventurer mode .

After removing stop words and punctuation

Dwarves singular Dwarf intelligent alcohol dependent humanoid creatures featured race fortress mode playable adventurer mode

Lexemes and lemmas

- A **lexeme** is a set of word forms sharing the same fundamental meaning.
  - word forms *run, runs, ran, running* ⟷ lexeme RUN
- A **lemma** is a wordform representing a given lexeme.
  - "dictionary form"; what you would put into a lexicon

```
>>> [token.lemma_ for token in nlp(text)]
['dwarf', 'be', '"', 'intelligent', '"', ',', 'alcohol', ...]
```

Lemmatization

Before lemmatization

Dwarves singular Dwarf intelligent alcohol dependent humanoid **creatures** featured race fortress mode playable adventurer mode

After lemmatization

dwarf singular **Dwarf** intelligent alcohol dependent humanoid **creature** featured race fortress mode playable adventurer mode

We might also want to consider lowercasing all tokens for some tasks.

Don’t apply these steps blindly!

Always consider both the task and model in order to decide which tokenization or preprocessing steps to use.

- Most preprocessing steps lose information!
  - e.g. lowercasing everything means searching for “dwarf” also finds “Dwarf”, but loses the distinction between “apple” *(the fruit)* and “Apple” *(the company)*
- Some models expect “raw” text as input, and perform tokenization etc. internally.
  - e.g. most modern large language models (LLMs)

Important concepts

- tokenization
- preprocessing
- stop words, stop word removal
- lemmatization

Ranked Retrieval

3

Ranked retrieval

- A **ranked retrieval** assigns scores to documents based on *how well* they match a given search query.
  - There are many possible ways of scoring!

- Search results can be ranked based on their score with respect to the query.
  - Can return a list of "top n documents".

Term weighting

- The score of a **document** d with respect to a **query** q is the sum of the weights of all **terms** t that occur in both d and q.

$$\text{score}(d, q) = \sum_{t \in (d \cap q)} \text{weight}(t, d)$$

terms that occur in both d and q

- Any specific way to assign weights to terms is called a **term weighting scheme**.

Term frequency

Example: Searching for "Moriarty"

Several Sherlock Holmes stories contain the term "Moriarty". We would like to rank stories that contain *many* occurrences higher than stories that contain only *few* occurrences.

- The **term frequency** of t in d is the number of times a term t occurs in a document d.
  - absolute frequency, count

$$\mathrm{tf}(t, d)$$

Background information about Moriarty…

Professor Moriarty's first appearance occurred in the 1983 short story *The Adventure of the Final Problem* [no. 23] […].

Holmes mentions Moriarty reminiscently in five other stories: *The Adventure of the Empty House* [no. 24], *The Adventure of the Norwood Builder* [no. 25], *The Adventure of the Missing Three-Quarter* [no. 34], *The Adventure of the Illustrious Client* [no. 45], and *His Last Bow* [no. 44].

Source: Wikipedia; numbering by Marco Kuhlmann.

Term frequency: “Moriarty”

expected	23	24	25	34	44	45
retrieved	23	24	25	34	44	45
# Moriarty	20	15	1	1	1	1

A problem with term frequency

Consider...

Is a document with *20 occurrences* of "Moriarty" really *20 times more relevant* than a document with only one occurrence?

- Intuitively, relevance is *not a linear function* of term frequency.
- **Log-frequency weighting** down-scales frequency using the log function:

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

Another problem with term frequency: "Moriarty Holmes"

expected	23	24	25	34	44	45
retrieved	28	36	48	22	25	52
# Moriarty	0	0	0	0	1	0
# Holmes	87	80	72	68	64	63

Document frequency

Example: Searching for "Moriarty Holmes"

- All Sherlock Holmes stories contain many instances of the term "Holmes".
- We would like to rank stories that *also* contain the term "Moriarty" higher.

- To implement this, we can look at the fraction of documents which contain that term, and then take the **inverse** of that.
  - The weight of a term should then grow proportionally to this inverse.

Inverse document frequency

- Let N be the total number of documents in the collection.
- The **document frequency** of t is the number of documents that contain the term t.

$$\mathrm{df}(t)$$

- The **inverse document frequency** is the multiplicative inverse of that:

$$\mathrm{idf}(t) = \log \frac{N}{\mathrm{df}(t)}$$

Term frequency-inverse document frequency

- The **tf–idf weight** of a term t in a document d is defined as:

$$\text{tf–idf}(t, d) = \text{tf}(t, d) \cdot \log \frac{N}{\text{df}(t)}$$

- **Variations** of this scheme exist; in scikit-learn, tf–idf is computed as:

$$\text{tf–idf}(t, d) = \text{tf}(t, d) \cdot \left(\log \frac{1 + N}{1 + \text{df}(t)} + 1\right)$$

  – Remember: N denotes the total number of documents in the collection.

Important concepts

- ranked retrieval
- term weighting scheme
- term frequency
- document frequency
- inverse document frequency
- tf–idf term weighting

The Vector Space Model

4

Reminder: Term-document matrix

	Scandal in Bohemia	Final problem	Empty house	Norwood builder	Dancing men	Retired colourman
Adair	0	0	1	0	0	0
Adler	1	0	0	0	0	0
Lestrade	0	0	1	1	0	0
Moriarty	0	1	1	1	0	0

- Documents as **sets of terms**: Is a term present in a document or not?

Term-document matrix with frequency values

	Scandal in Bohemia	Final problem	Empty house	Norwood builder	Dancing men	Retired colourman
Adair	0	0	14	0	0	0
Adler	13	0	0	0	0	0
Lestrade	0	0	10	51	0	0
Moriarty	0	20	15	1	0	0

- Documents as **bags of terms**: *How often* is a term present in a document?

Term-document matrix with tf-idf values

	Scandal in Bohemia	Final problem	Empty house	Norwood builder	Dancing men	Retired colourman
Adair	0.0000	0.0000	0.0692	0.0000	0.0000	0.0000
Adler	0.0531	0.0000	0.0000	0.0000	0.0000	0.0000
Lestrade	0.0000	0.0000	0.0291	0.1424	0.0000	0.0000
Moriarty	0.0000	0.0845	0.0528	0.0034	0.0000	0.0000

- Documents as bags of terms, but now **weighted** by tf–idf.

Turning documents into vectors

Idea #1

Represent **documents as vectors** in a high-dimensional space.

- **Dimensions** *(axes)* of the space correspond to terms in the **vocabulary**.
  - Could be: set of all words in the collection; set of most frequent words; …
- Values of the vector depend on the **term weighting scheme**, e.g. counts, tf–idf, …
  - In scikit-learn: CountVectorizer, TfidfVectorizer

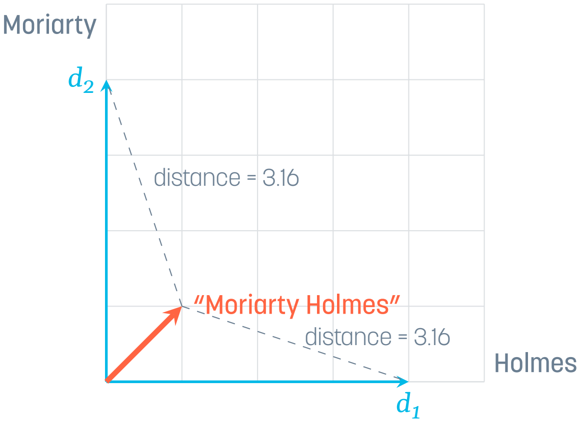
Turning queries into vectors

Idea #2

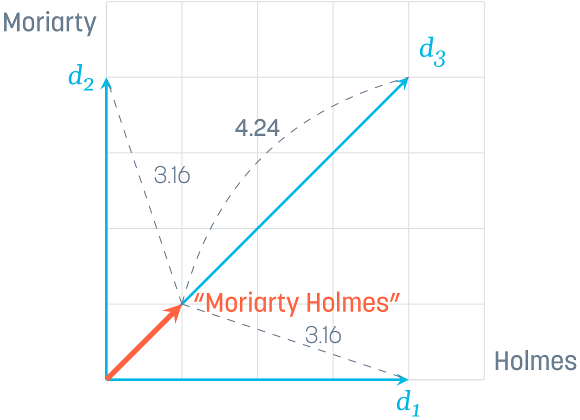
Represent **queries as vectors** in the *same* vector space.

- We can use linear algebra to compute **similarity** between vectors.
  - similarity = proximity in the vector space
- To **score a candidate document**, we compute the similarity between its document vector and the query vector.

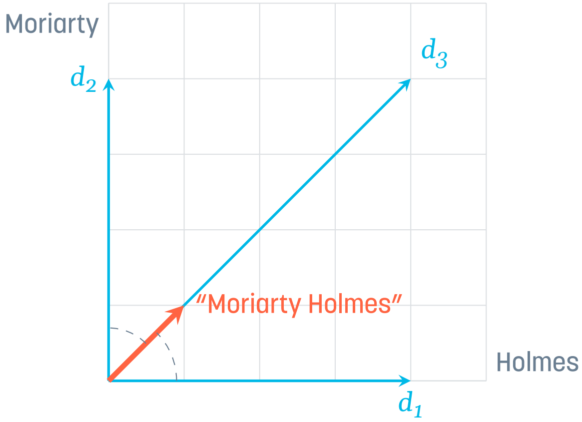
Euclidean distance



Euclidean distance: A problem



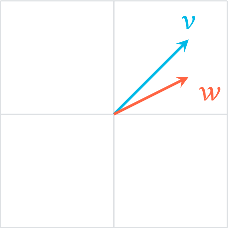
From distances to angles



- Euclidean distance also **varies with length** of the vectors.
  - Vectors with similar distributions can have very different lengths.
- We should rank documents based on the **angle** between vectors instead.
  - Turns out this also has computational benefits!

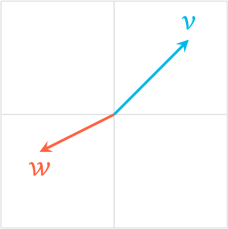
The dot product

$$v = (+2, +2)$$
$$w = (+2, +1)$$



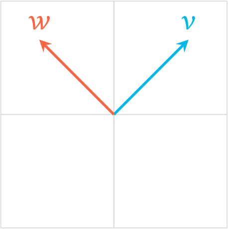
$$v \cdot w = +6$$

$$v = (+2, +2)$$
$$w = (-2, -1)$$



$$v \cdot w = -6$$

$$v = (+2, +2)$$
$$w = (-2, +2)$$



$$v \cdot w = \pm 0$$

Cosine similarity

- The dot product is still sensitive to length. → **normalize** each vector to unit length

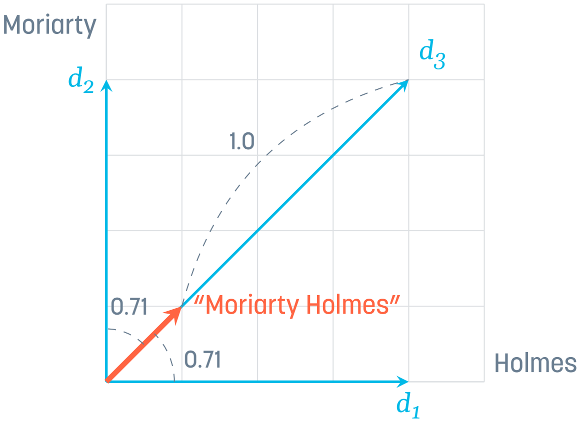
- The **cosine similarity** of two vectors is the length-normalized dot product:

$$\cos(v, w) = \frac{v}{|v|} \cdot \frac{w}{|w|} = \frac{v \cdot w}{|v||w|}$$

$$= \frac{\sum_{i=1}^{d} v_i w_i}{\sqrt{\sum_{i=1}^{d} {v_i}^2}\sqrt{\sum_{i=1}^{d} {w_i}^2}}$$

- Cosine similarity ranges from **-1** (opposite) to **+1** (identical).

Cosine similarity: A problem solved



Important concepts

- Euclidean distance
- dot product
- cosine similarity

How to evaluate IR systems?

5

Evaluation of IR systems

To evaluate an IR system, we need:

- a document collection
- a collection of queries
- a **gold-standard judgement** of relevance
  - Gold standard: the “best available” test or benchmark
  - Typically produced by human annotators

Producing a gold-standard relevance judgement

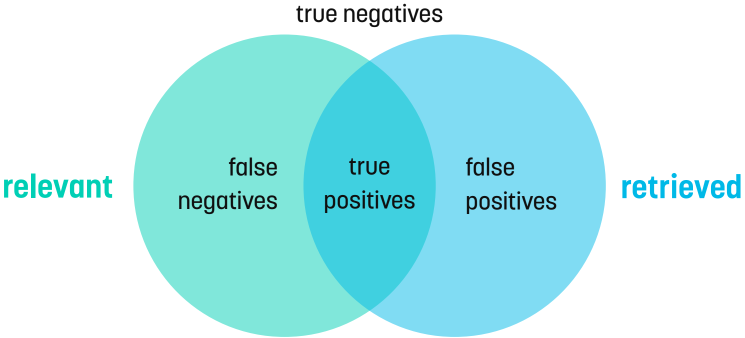
Number	0
Title	ibuprofen COVID-19
Description	Can ibuprofen worsen COVID-19?
Answer	No
Evidence	`https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7287029`
Narrative	Ibuprofen is an anti-inflammatory drug used to reduce fever and treat pain and inflammation. Recently, there has been a debate over whether Ibuprofen can worsen the effects of COVID-19. A helpful document might explain in clear language that there is no scientific evidence supporting this concern. A harmful document might create anxiety and/or cause people to avoid taking the drug.

Sample taken from TREC 2020

Gold-standard relevance judgments

query	document 1	document 2	document 3
505	✓	✓	✗
506	✗	✗	✗
507	✓	✗	✗
508	✗	✗	✓
509	✓	✓	✗
510	✓	✓	✓
511	✗	✗	✗

Precision and recall for Boolean retrieval



$$P = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{retrieved}|}$$

$$R = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{relevant}|}$$

F1-measure

- A good system should **balance** between precision and recall.
- The **F1-measure** is the harmonic mean of the two values:

$$\text{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Evaluation of ranked retrieval

rank	document	relevant?
1	191	✓
2	153	✓
3	28	✓
4	198	✓
5	174	✗
6	178	✗
7	145	✗
8	183	✗

best

rank	document	relevant?
1	191	✓
2	174	✗
3	153	✓
4	178	✗
5	28	✓
6	145	✗
7	198	✓
8	183	✗

worse

Evaluation of ranked retrieval

Intuition

A good system ranks relevant documents high, and irrelevant documents low.

- We can compute precision and recall *at different ranks*.
  - This generalizes the evaluation of Boolean retrieval to ranked retrieval.
- In practice, recall is hard to evaluate!
  - Requires us to know *all* relevant documents.
- Evaluation tends to **focus on precision**.

Mean Average Precision (MAP)

1. For each query, compute **precision up to each rank** where a relevant document was returned.
   – up to a fixed maximal rank, say $k = 100$
2. **Average the precision values** for each specific query.
3. **Average the averages** for all queries in the collection used for the evaluation.
   – macro-averaging: each query counts equally

Average precision for the “best” example

rank	document	relevant?	precision @ rank
1	191	✓	1/1
2	153	✓	2/2
3	28	✓	3/3
4	198	✓	4/4
5	174	✗	
6	178	✗	
7	145	✗	
8	183	✗	

- Average precision for this query:

$$\frac{\frac{1}{1}+\frac{2}{2}+\frac{3}{3}+\frac{4}{4}}{4} = 1.00$$

Average precision for the “worse” example

rank	document	relevant?	precision @ rank
1	191	✔	1/1
2	174	✘	
3	153	✔	2/3
4	178	✘	
5	28	✔	3/5
6	145	✘	
7	198	✔	4/7
8	183	✘	

- Average precision for this query:

$$\frac{\frac{1}{1} + \frac{2}{3} + \frac{3}{5} + \frac{4}{7}}{4} \approx 0.71$$

Important concepts

- precision
- recall
- F1-score
- mean average precision (MAP)

LINKÖPING
UNIVERSITY