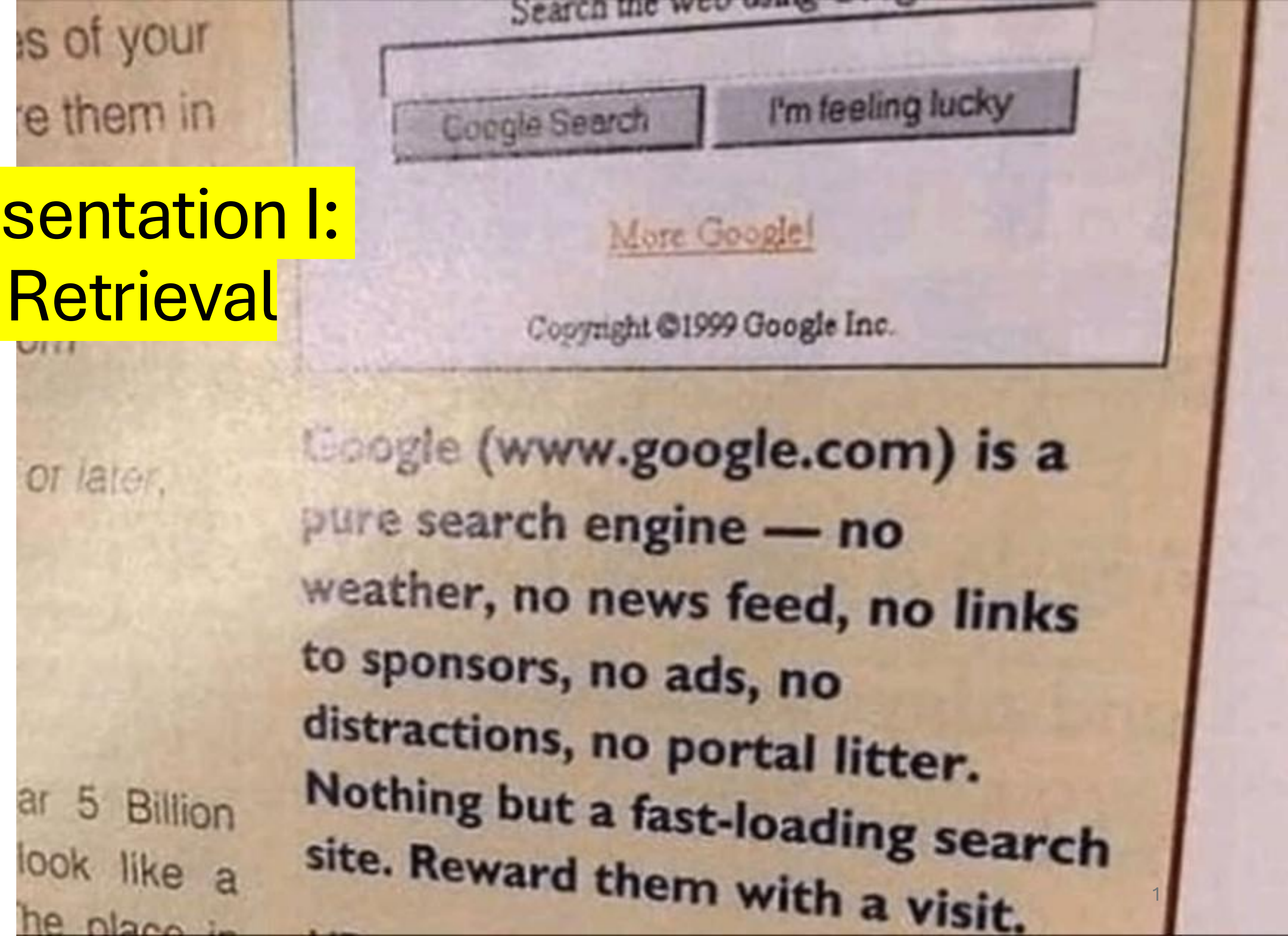
Word Representation I: Information Retrieval

Lecture 3



Representing Data

- Earlier success in computer vision.
  - Navlab 5 (Jochem et al., 1995)

- Much more intuitive to convert images into vector representations.

Representing Data

- Numeric Data:
  - E.g. credit score:

Monthly Income	Number Of Open Credit Lines And Loans	Number Of Times 90 Days Late	Number Real Estate Loans Or Lines	Number Of Time 60-89 Days Past Due Not Worse	Number Of Dependents
9120	13	0	6	0	2
2600	4	0	0	0	1
3042	2	1	0	0	0
3300	5	0	0	0	0
63588	7	0	1	0	0
3500	3	0	1	0	1
NA	8	0	3	0	0
3500	8	0	0	0	0

Give Me Some Credit (gmsc): https://www.kaggle.com/c/GiveMeSomeCredit

Representing Data

- Numeric Data:
  - E.g. credit score:
- Images:
  - Gray scale or RGB

MNIST dataset
Handwritten numbers

- Videos:
  - Images on a timeline

How can we represent words (tokens) with vectors?

Not as straightforward, hmmm

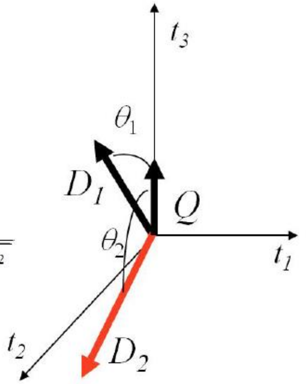
Everything will make sense after the next lecture

Early Success

- WSD Bag of Words
- $(T_1, T_2, T_3 \ldots)$

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\text{CosSim}(\boldsymbol{d_j}, \boldsymbol{q}) = \frac{\vec{d}_j \cdot \vec{q}}{\left|\vec{d}_j\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim$(D_1, Q) = 10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim$(D_2, Q) = 2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$
$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

- Information Retrieval
  - TF-IDF
  - BM25
  - LSA
- IR Evaluation

Today

IR – Introduction, Evaluation

1 **Introduction**

- Inverted Index
- Search & Relevance
- TF-IDF & BM25
- LSA

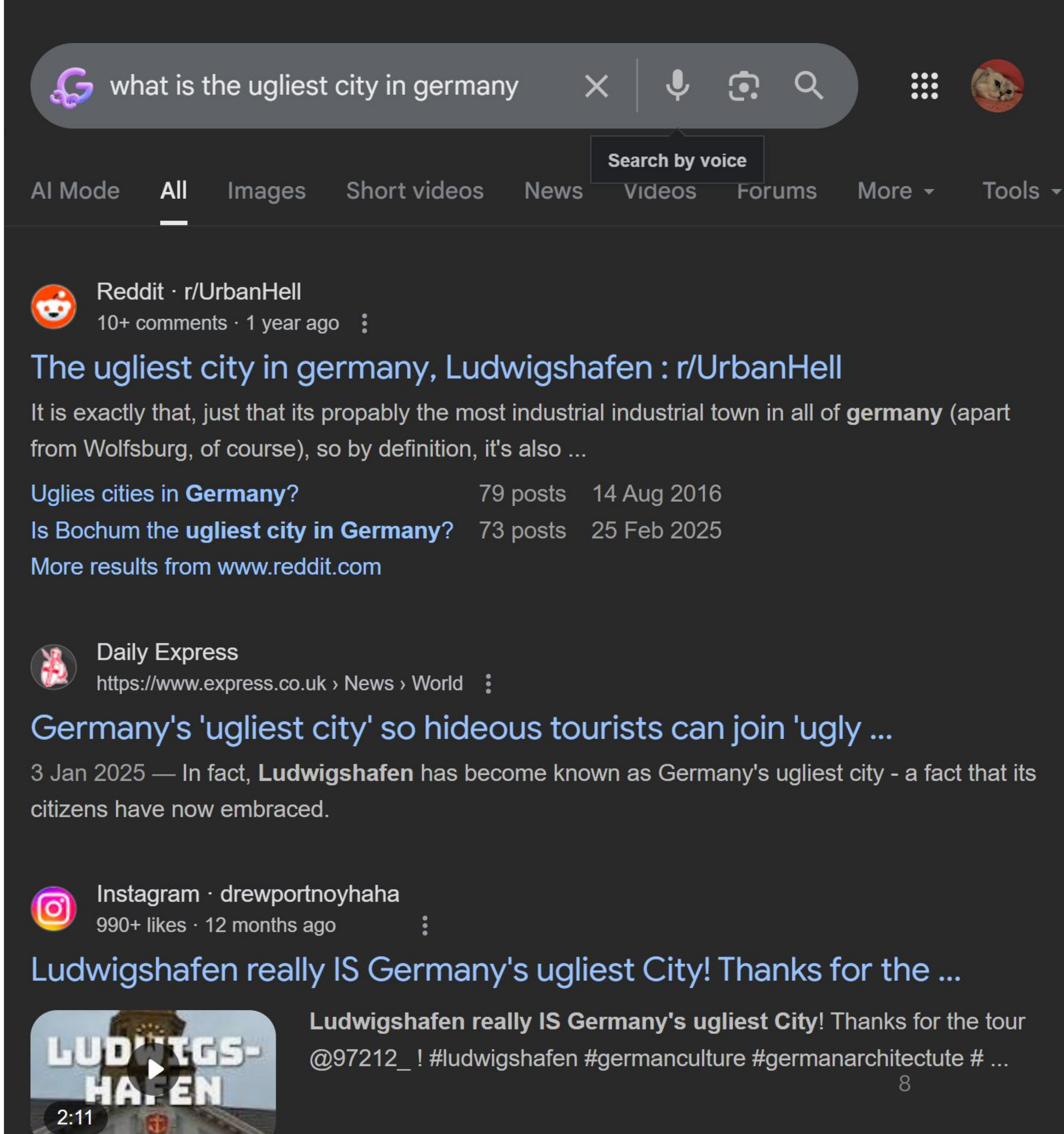
2 Evaluation

- Precision & Recall
- MRR & MAP
- nDCG

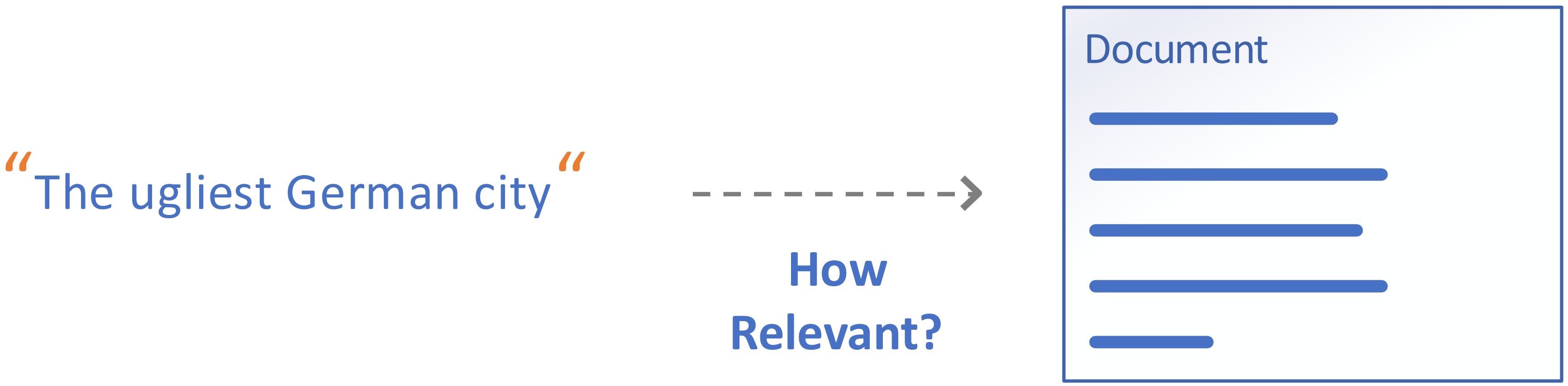
Based on: Lecture 1 - Foundations of IR + Lecture 3 - Relevance & Scoring
https://github.com/sebastian-hofstaetter/teaching/tree/master/introduction-to-information-retrieval/lectures

Information Retrieval

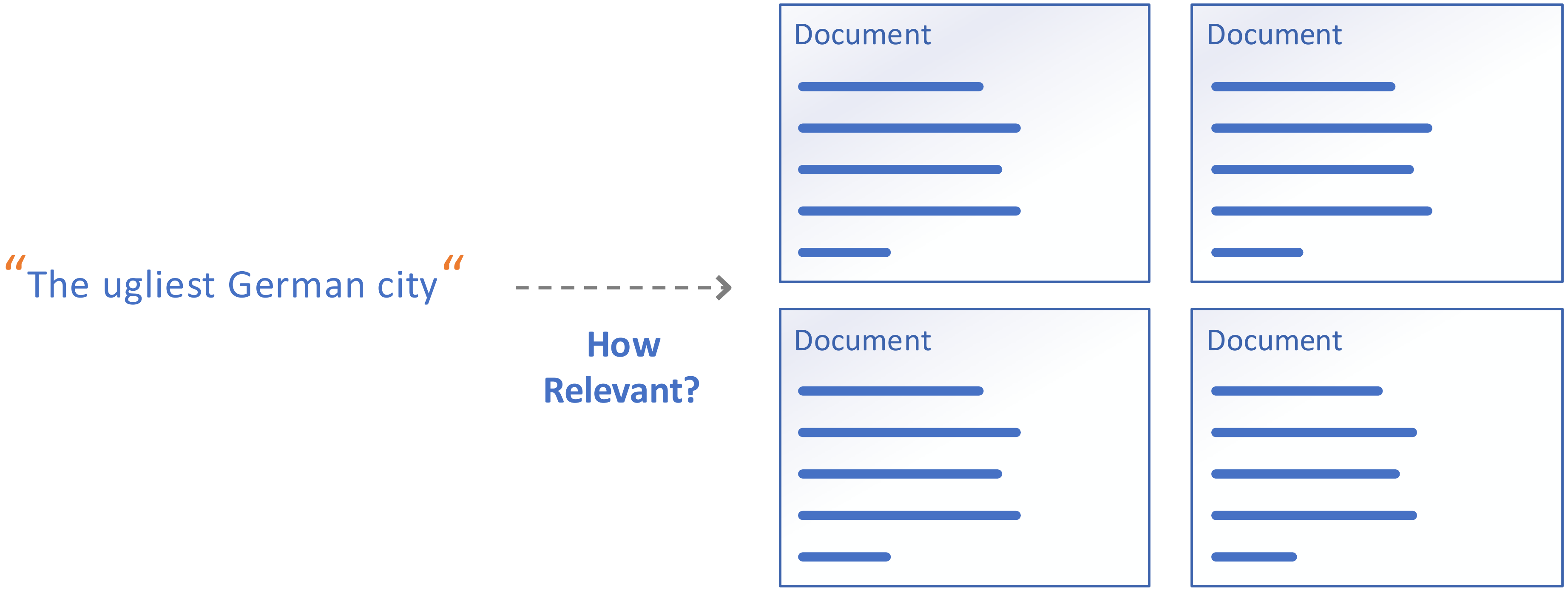
- Basically, Google Search.
- Tons of documents.
- 1 Search query.



Information Retrieval



Information Retrieval (finding the needle in the haystack)



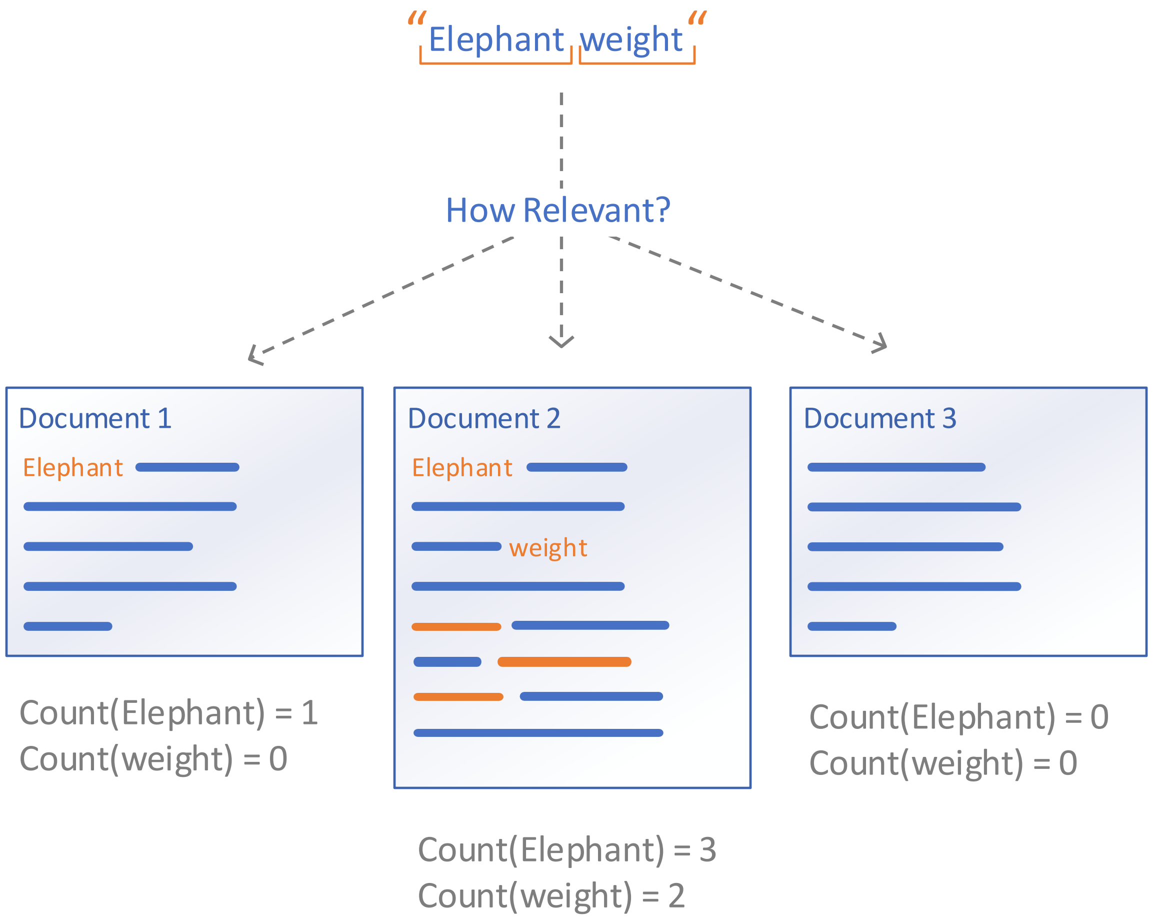
Notes on terminology

- **Documents** can be anything: a web page, word file, text file, article …
  (we assume it to be text for the moment)
  - A lot of details to look out for: encoding, language, hierarchy, fields, …

- **Collection**: A set of documents (we assume it to be static for the moment)

- **Relevance**: Does a document satisfy the information need of the user and does it help complete the user's task?

Relevance (based on text content)



- If a word appears more often -> more relevant
- Solution: count the words
- If a document is longer, words will tend to appear more often -> take into account the document length
- Counting only when we have a query is inefficient

Today

IR – Introduction, Evaluation

1 Introduction

- **Inverted Index**
- Search & Relevance
- TF-IDF & BM25
- LSA

2 Evaluation

- Precision & Recall
- MRR & MAP
- nDCG

Inverted Index

- Inverted index allows to efficiently retrieve documents from large collections
- Inverted index stores all statistics per term (that the scoring model needs)
  - **Document frequency:** how many documents contain the term
  - **Term frequency per document:** how often does the term appear per document
  - Document length
  - Average document length
- Save statistics in a format that is accessible **by a given term**
- Save metadata of a document (Name, location of the full text, etc..)

Inverted Index

Document data

```
Document Ids & Metadata:

[0] = (”Wildlife”, ”location”,...)
[1] = (”Zoo Vienna” ,...)
...

Document Lengths:

[0] = 231 [1] = 381 ...
```

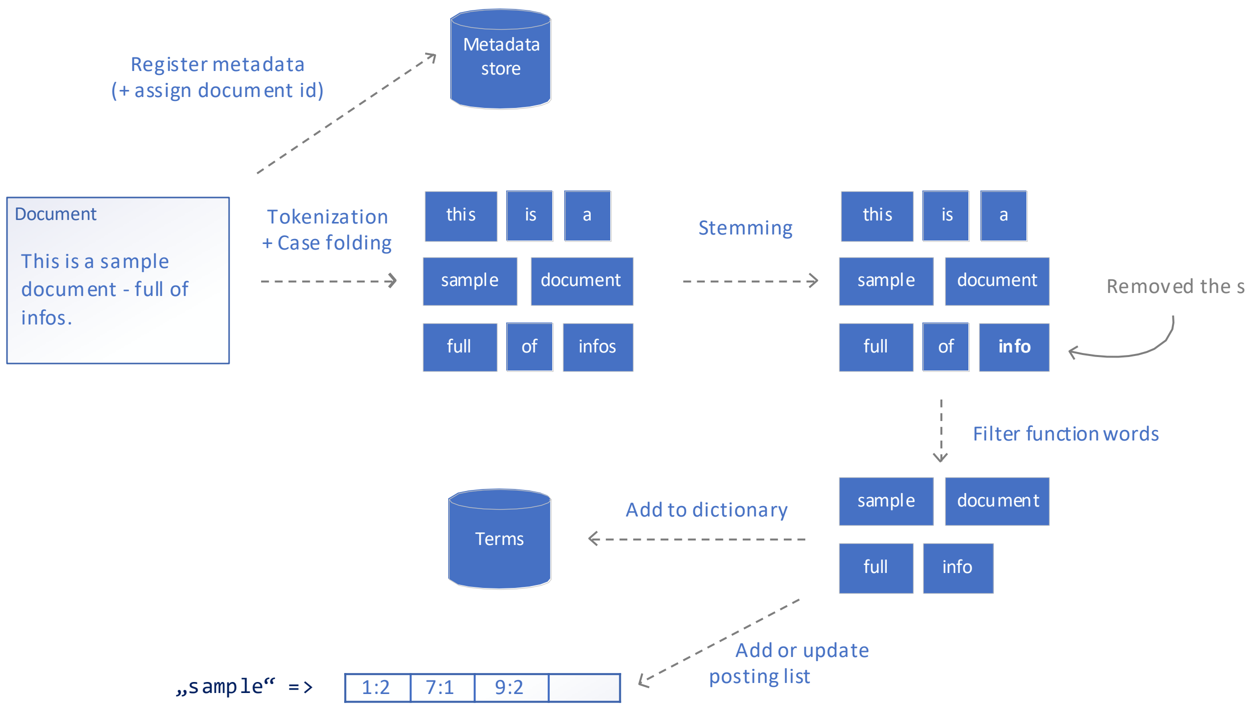
Term data

```
”elephant“ => | 1:5 | 2:1 | 3:5 | 4:5 | … |

”lion“ =>     | 1:2 | 7:1 | 9:2 | … |

”weight“ =>   | 4:1 | 6:4 | … |

...
```

DocId Term Frequency

- Every document gets an internal document id
- Term dictionary is saved as a search friendly data structure (more on that later)
- Term Frequencies are stored in a “posting list” = a list of doc id, frequency pairs

Creating the Inverted Index



- Simplified example pipeline
- Linguistic models are language dependent
- A query text and a document text both have to undergo the same steps

Today

IR – Introduction, Evaluation

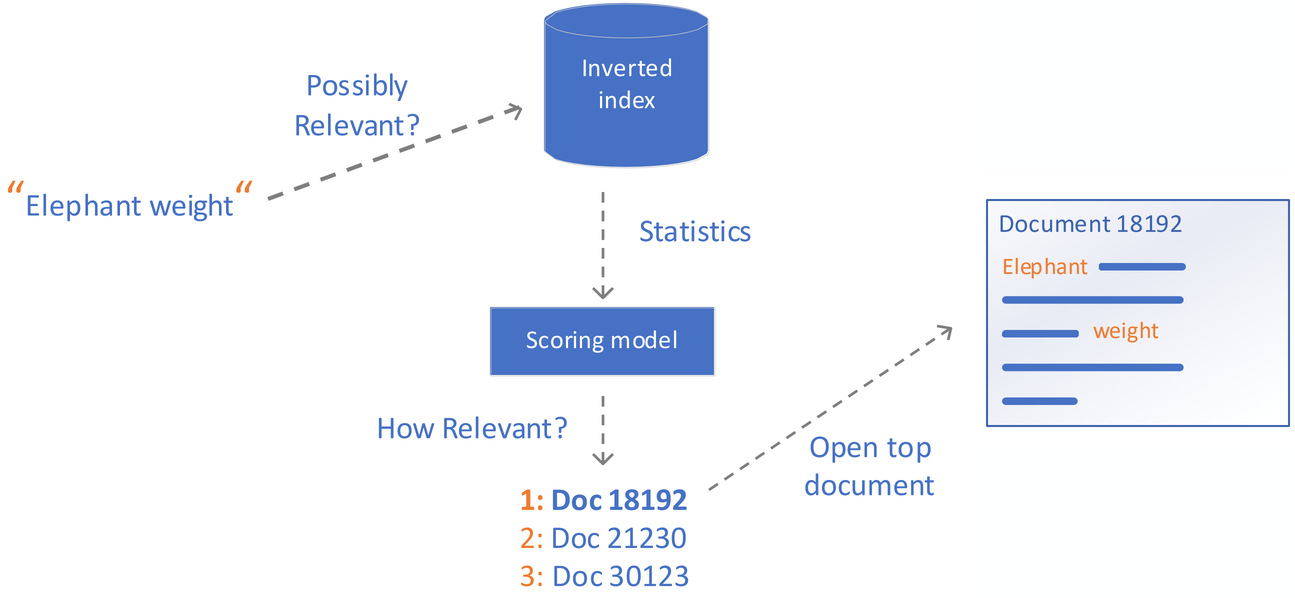
1 Introduction

- Inverted Index
- **Search & Relevance**
- TF-IDF & BM25
- LSA

2 Evaluation

- Precision & Recall
- MRR & MAP
- nDCG

Querying the Inverted Index



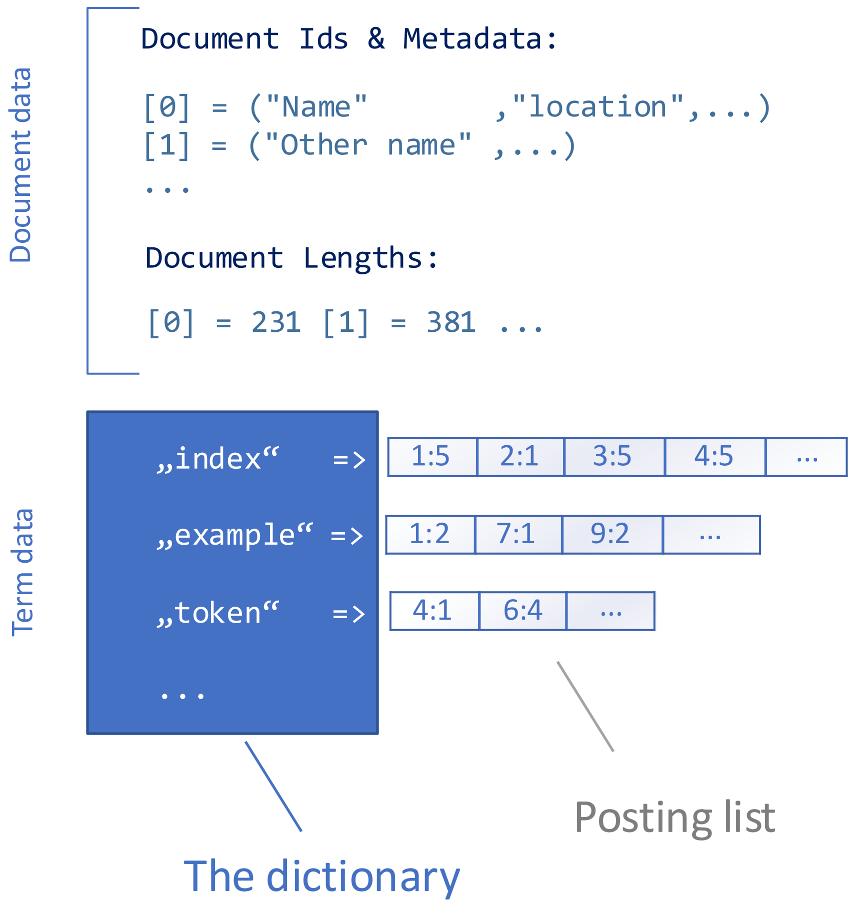
- No need to read full documents
- Only operate on frequency numbers of potentially relevant documents*
- Sort documents based on relevance score – retrieve most relevant documents

* it's not that easy because a document could be relevant without containing the exact query terms – but for now keep it simple

Types of queries (including, but not limited to)

- **Exact matching**: match full words and concatenate multiple query words with “or”
- **Boolean queries**: “and” / “or” / “not” operators between words
- **Expanded queries**: automatically incorporate synonyms and other similar or relevant words into the query
- **Wildcard queries**, **phrase queries, phonetic queries** (e.g. Soundex) …

Inverted Index: Dictionary



- Dictionary<T> maps text to T
  - T is a posting list or potentially other data about the term depending on the index
- Wanted properties:
  - Random lookup
  - Fast (creation & especially lookup)
  - Memory efficient (keep the complete dictionary in memory)
- Naturally, there are a lot of choices

Scoring model

- Input: statistics, Output: floating point value (i.e. the score)
- Evaluated pairwise – 1 query, 1 document: $score\ (q, d)$
- Capture the notion of relevance in a mathematical model

Today we focus on free-text queries & „ad-hoc“ document retrieval

(document content only)

Search algorithm

float *Scores*={}

for each query term *q*

fetch posting list for *q*

for each pair(d, $tf_{t,d}$) in posting list

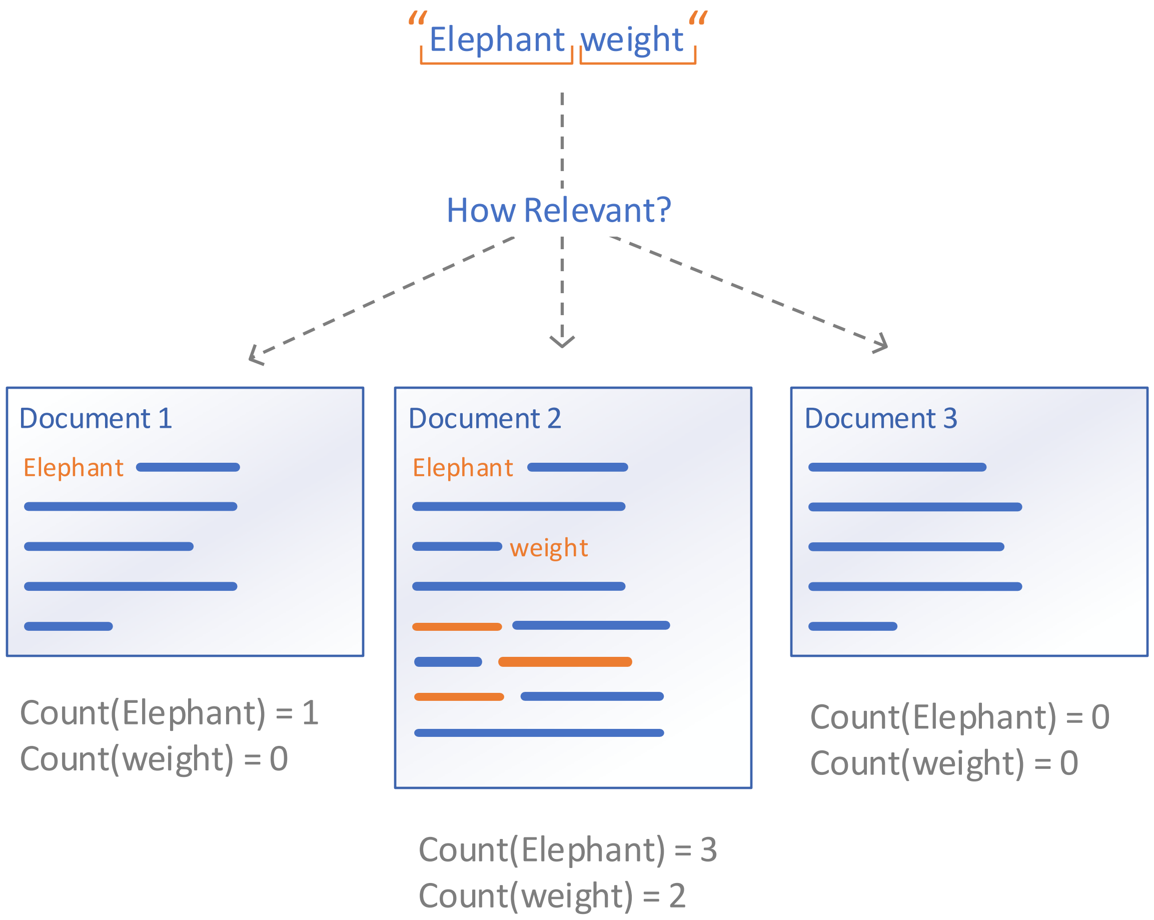
if *d* not in *Scores* **do** *Scores*[*d*]=0

Scores[*d*] += score(q, d, $tf_{t,d}$, …)

return Top *K* entries of *Scores*

We transform information back to a document centric view (from the term centric view in the inverted index)

Relevance



- If a word appears more often → more relevant
- Solution: **count the words**
- If a document is longer, words will tend to appear more often → take into account the document length

Relevance limitations

- “Relevance” means relevance to the need rather than to the query
  - “Query” is shorthand for an instance of information need, its initial verbalized presentation by the user
- Relevance is assumed to be a binary attribute
  - A document is either relevant to a query/need or it is not
- We need these oversimplifications to create & evaluate mathematical models

From: A probabilistic model of information retrieval: development and comparative experiments, Spärck Jones et al. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.134.6108&rep=rep1&type=pdf

Today

IR – Introduction, Evaluation

1 Introduction

- Inverted Index
- Search & Relevance
- **TF-IDF & BM25**
- LSA

2 Evaluation

- Precision & Recall
- MRR & MAP
- nDCG

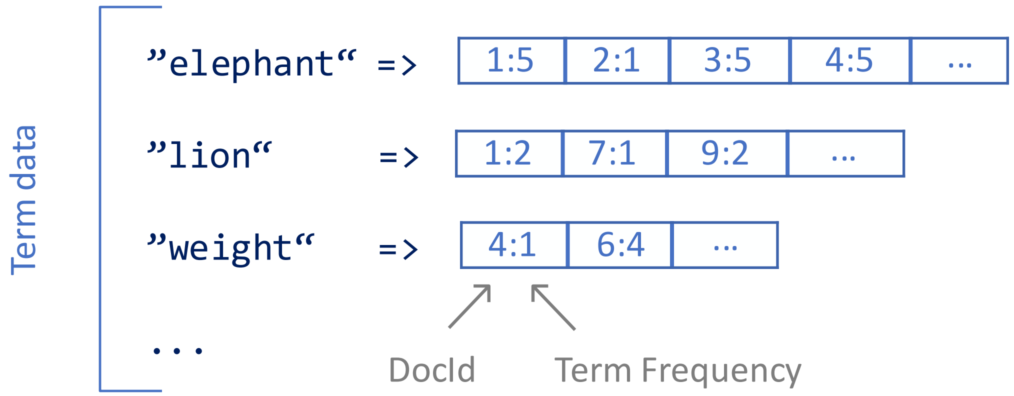
Term Frequency – conceptional data view

- **Bag of words**: word order is not important
- First step for a retrieval model: number of occurrences counts!
- $tf_{t,d}$ number of occurrences of term *t* in document *d*

		Documents					
		Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Terms	Antony	157	73	0	0	0	0
	Brutus	4	157	0	1	0	0
	Caesar	231	227	0	2	1	1
	Calpurnia	0	10	0	0	0	0
	Cleopatra	57	0	0	0	0	0
	mercy	2	0	3	5	5	1
	worser	2	0	1	1	1	0

Term Frequency – actual data storage

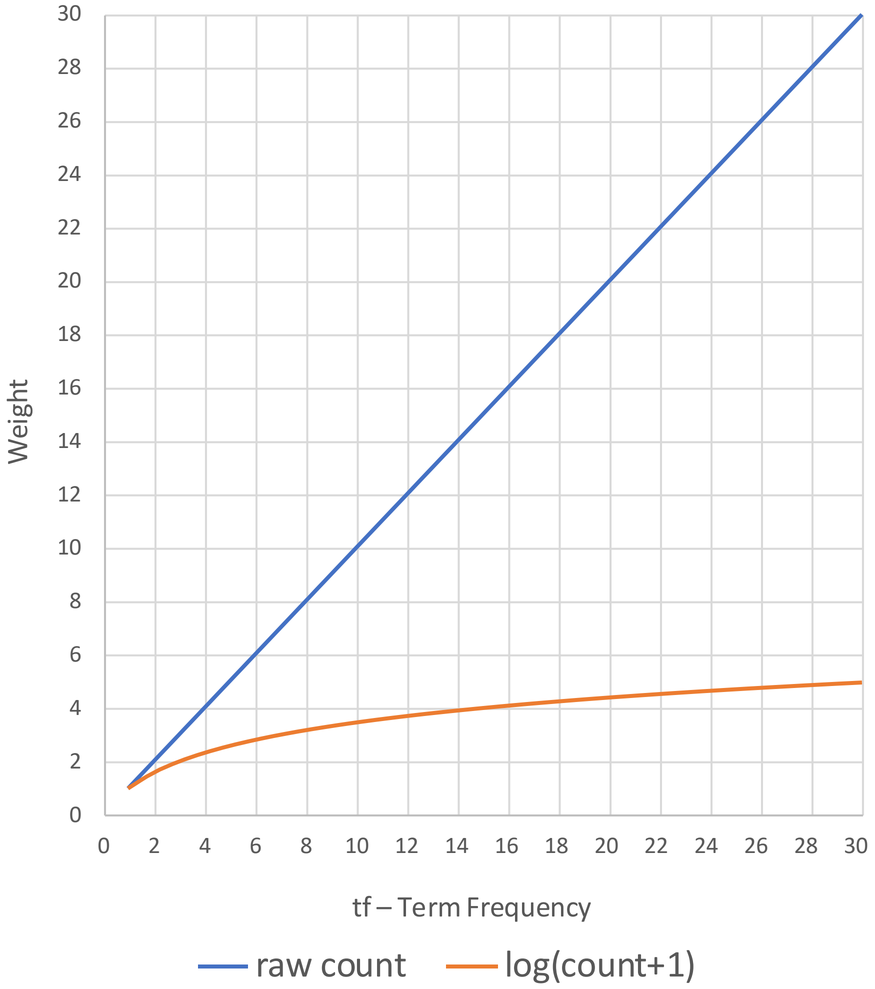
- Inverted index saves only non-0 entries, not the whole matrix
  - Otherwise we would waste a lot of storage capacity
- Therefore not good at random lookups into the document column
  - Needs to iterate through the posting list to find the correct document
  - However, for scoring models $tf_{t,d}$ with 0 can be skipped



TF - Term Frequency

- $tf_{t,d}$ = how often does term t appear in document d
- Powerful starting point for many retrieval models
- Main point of our intuition at the beginning
- Using the raw frequency is not the best solution
  - Use relative frequencies
  - Dampen the values with logarithm

Term Frequency & Logarithm



- In long documents, a term may appear hundred of times.
- Retrieval experiments show that using the logarithm of the number of term occurrences is more effective than raw counts.
- Commonly used approach: apply logarithm

$$\log(1 + tf_{t,d})$$

Document Frequency

- df_t = in how many documents does term t appear in

- Rare terms are more informative than frequent terms
  - Recall function words (and, or, the, …)
- Consider a term in the query that is rare in the collection
  - e.g., *Darmstadt in a news corpora*
- A document containing this term is very likely to be relevant to the query *TU Darmstadt*

→ We want a high weight for rare terms like *Darmstadt*.

IDF – Inverse Document Frequency

- A common way of defining the inverse document frequency of a term is as follows:

$$\text{idf}(t) = log\,\frac{|D|}{df_t}$$

- df_t is an inverse measure of the "informativeness" of the term
- $df_t \leq |D|$
- Logarithm is used also for *idf* to "dampen" its effect.

$\lvert D \rvert$	Total # of documents
df_t	# of Documents with $tf_{t,} > 0$

TF-IDF

$$TF\_IDF(q,d) = \sum_{t \in T_d \cap T_q} \underbrace{log(1 + tf_{t,d})}_{\text{increases with the number of occurrences within a document}} * \underbrace{log(\frac{|D|}{df_t})}_{\text{increases with the rarity of the term in the collection}}$$

$\sum_{t \in T_d \cap T_q}$	Sum over all query terms, that are in the index
$tf_{t,d}$	Term frequency
$\|D\|$	Total # of documents
df_t	# of Documents with $tf_{t,d} > 0$

- A rare word (in the collection) appearing a lot in one document creates a high score
- Common words are downgraded

For more variations: https://en.wikipedia.org/wiki/Tf-idf

TF-IDF – Usage

- Useful not only as a standalone model in document retrieval

- Weights used as a base for many other retrieval models
  - Example: Vector Space Model (VSM) works better with tf-idf weights

- Also useful as a generic word weighting mechanism for NLP
  - Task agnostic importance of a word in a document in a collection
  - Assign every word in a collection its tf-idf score

Example

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import torch
from torch.nn.functional import cosine_similarity as sim

if __name__ == '__main__':
    corpus = ['I did not hit her',
              'I did not',
              'Oh hi Mark']

    tfidf = TfidfVectorizer(stop_words='english')
    x = torch.tensor(tfidf.fit_transform(corpus).todense())
    fs = tfidf.get_feature_names_out()
    print(fs)
    print(x)

    print(corpus[0], corpus[1], sim(x[0], x[1], dim=0), sep='\t')
    print(corpus[0], corpus[2], sim(x[0], x[2], dim=0), sep='\t')
    print(corpus[1], corpus[2], sim(x[1], x[2], dim=0), sep='\t')

    print(fs[0], fs[2], sim(x[:,0], x[:,2], dim=0), sep='\t')
    print(fs[1], fs[2], sim(x[:,1], x[:,2], dim=0), sep='\t')
```

Result


```
corpus = ['I did not hit her',
          'I did not',
          'Oh hi Mark']
tfidf = TfidfVectorizer(stop_words='english')
x = torch.tensor(tfidf.fit_transform(corpus).todense())
fs = tfidf.get_feature_names_out()
print(fs)
print(x)
print(corpus[0], corpus[1], sim(x[0], x[1], dim=0), sep='\t')
print(corpus[0], corpus[2], sim(x[0], x[2], dim=0), sep='\t')
print(corpus[1], corpus[2], sim(x[1], x[2], dim=0), sep='\t')
print(fs[0], fs[2], sim(x[:,0], x[:,2], dim=0), sep='\t')
print(fs[1], fs[2], sim(x[:,1], x[:,2], dim=0), sep='\t')
```

```
['did' 'hi' 'hit' 'mark' 'oh']
[[0.60534851 0.         0.79596054 0.         0.        ]
 [1.         0.         0.         0.         0.        ]
 [0.         0.57735027 0.         0.57735027 0.57735027]]
I did not hit her	I did not	0.6053485081062917
I did not hit her	Oh hi Mark	0.0
I did not	Oh hi Mark	0.0
did	hit	0.5178561161676976
hi	hit	0.0
```

BM25

- Created 1994 by Robertson et al.
- Grounded in probabilistic retrieval

- In general, BM25 improves on TF-IDF results

- But only set as a default scoring in Lucene in 2015

Original paper: http://www.staff.city.ac.uk/~sb317/papers/robertson_walker_sigir94.pdf

TF-IDF vs BM25 in Lucene https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/

BM25 (as defined by Robertson et al. 2009)

$$BM25(q,d) = \sum_{t \in T_d \cap T_q} \frac{tf_{t,d}}{k_1((1-b) + b\frac{dl_d}{avgdl}) + tf_{t,d}} * log \frac{|D| - df_t + 0.5}{df_t + 0.5}$$

- Simpler than the original formula
  - Over time it was shown that more complex parts not needed

Details (a lot of them): The Probabilistic Relevance Framework: BM25 and Beyond
http://www.staff.city.ac.uk/~sb317/papers/foundations_bm25_review.pdf

$\sum_{t \in T_d \cap T_q}$	Sum over all query terms, that are in the index
$tf_{t,d}$	Term frequency
dl_d	Document length
$avgdl$	Average document length in index
$\|D\|$	Total # of documents
df_t	# of Documents with $tf_{t,} > 0$
k_1, b	Hyperparameters

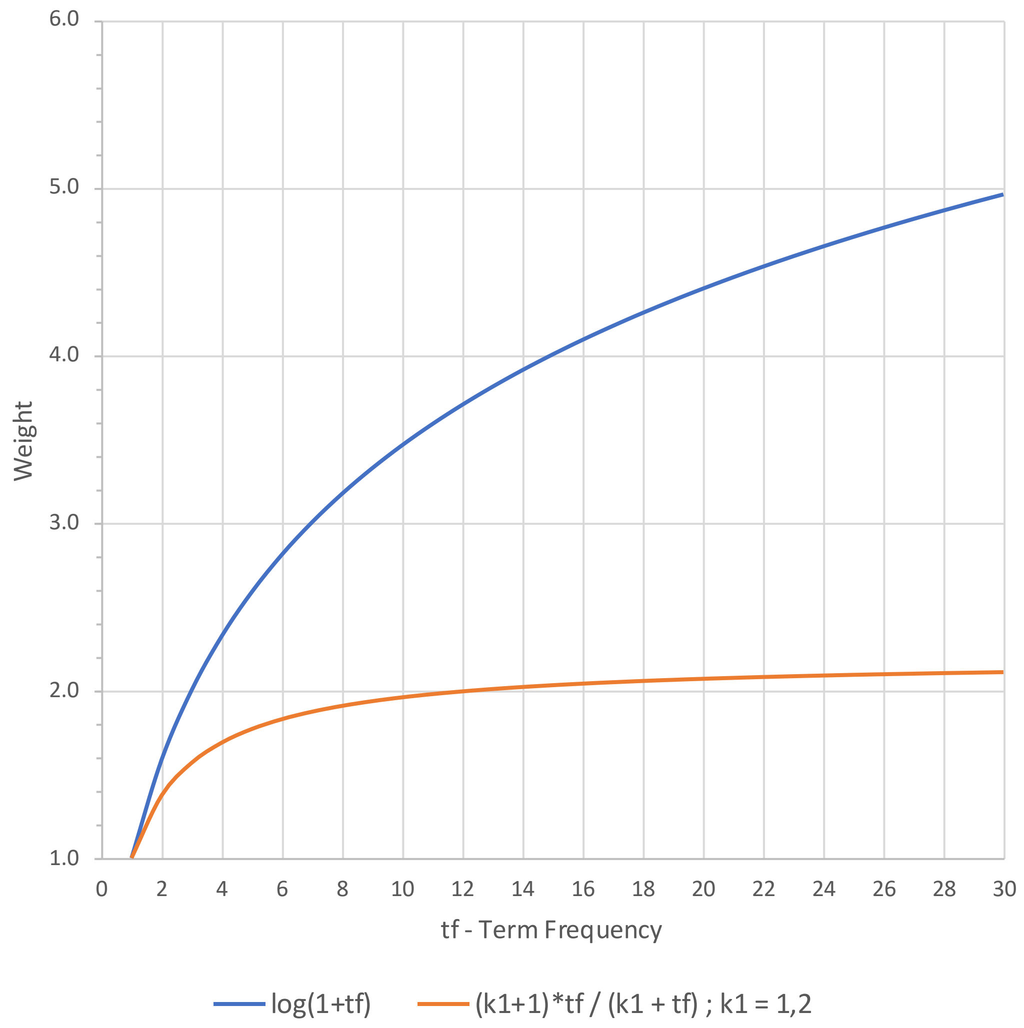
BM25 vs. TF-IDF

- Simple case of BM25 looks a lot like TF-IDF

- 1 main difference: BM25 tf component contains saturation function
  - Therefore works better in practice

- BM25 variants can be adapted to:
  - Incorporate additional reference information
  - Long(er) queries
  - multiple fields

BM25 vs. TF-IDF - Saturation



- **TF-IDF:** weight is always increasing (even with log)
- **BM25:** diminishing returns quickly = asymptotically approaches $k_1 + 1$

Note: we added $(k_1{+}1)$ to the numerator to make tf@1 = 1, but it does not change the ranking because it is added to every term

Note: we assume the doc length = avgdl

BM25 vs. TF-IDF - Example

- Suppose your query is “machine learning”
- Suppose you have 2 documents with term counts:
  - doc1: learning 1024; machine 1
  - doc2: learning 16; machine 8

- TF-IDF: $\log(\mathrm{tf}) * \log(|D|/\mathrm{df})$
  - doc1: $11 * 7 + 1 * 10 = \mathbf{87}$
  - doc2: $5 * 7 + 4 * 10 = \mathbf{75}$

- BM25: $k_1 = 2$
  - doc1: $3 * 7 + 1 * 10 = \mathbf{31}$
  - doc2: $2.67 * 7 + 2.4 * 10 = \mathbf{42.7}$

Example taken from: https://web.stanford.edu/class/cs276/

Hyperparameters

- k_1, b are hyperparameters = they are set by us, the developers

- k_1 controls term frequency scaling
  - k_1 = 0 is binary model; k_1 large is raw term frequency
- b controls document length normalization
  - b = 0 is no length normalization; b = 1 is relative frequency (fully scale by document length)

- Common ranges: 0.5 < b < 0.8 and 1.2 < k_1 < 2

Summary: Part 1

1. We save statistics about terms in an inverted index

2. The statistics in the index can be access by a given term (query)

3. TF-IDF & BM25 use term and document frequencies to score a query & doc

Today

IR – Introduction, Evaluation

1 Introduction

- Inverted Index
- Search & Relevance
- TF-IDF & BM25
- **LSA**

2 Evaluation

- Precision & Recall
- MRR & MAP
- nDCG

Latent Semantic Analysis (LSA)

- TF-IDF is actually pretty stupid:
  - Car vs cars → lemmatisation, wordnet…
  - Car vs automobile → different tokens!
  - No generalisation

- TF-IDF is **very sparse**:
  - We need to keep track of an **M x N** matrix of token frequencies
    - M: documents, N: vocab size
  - Say 10,000 words, 10,000 documents: 100M values
  - BERT & GPT2: 410M parameters

Latent Semantic Analysis (LSA)

- Basic idea: use **singular value decomposition (SVD)** to encourage generalization.

LSA: Count!

- Factorize a (maybe weighted, maybe log scaled) term-document or word-context matrix (Schütze 1992) into $U\Sigma V^T$
  - Singular value decomposition (SVD)
- Retain only *k* singular values, in order to generalize.

Latent Semantic Analysis (LSA)

- Step 1: Build a Term–Document Matrix

	Doc 1	Doc 2	Doc 3
cat	2	0	1
dog	1	3	0
apple	0	0	2

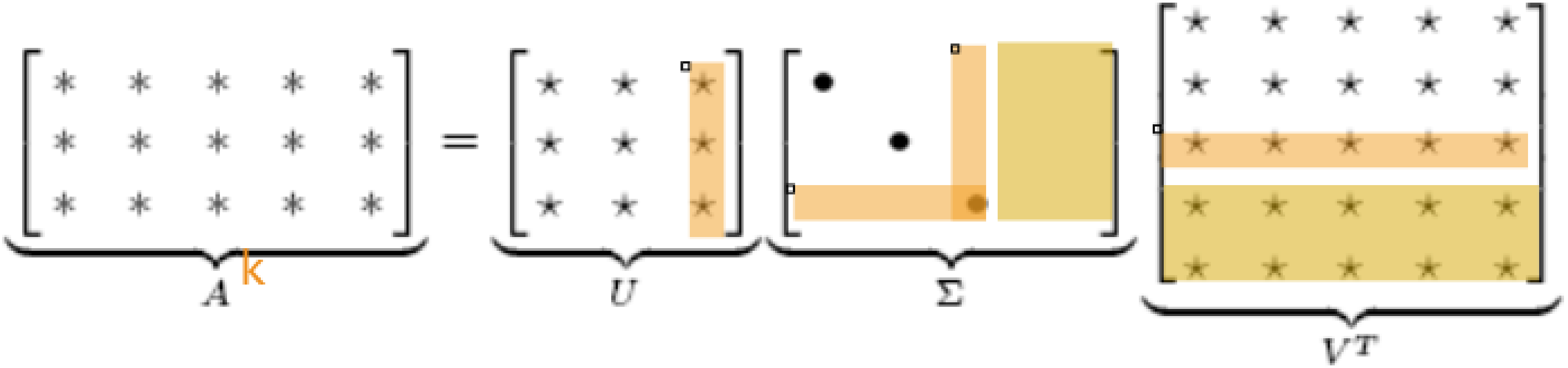
- Step 2: Apply Singular Value Decomposition (SVD)
  - **U**: word embeddings
  - **V**: document embeddings
  - **Σ**: diagonal matrix of singular values (importance of each latent dimension)

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

Latent Semantic Analysis (LSA)

- Step 3: Reduce Dimensionality
  - Keep only the top k singular values and columns of U and V:

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$$

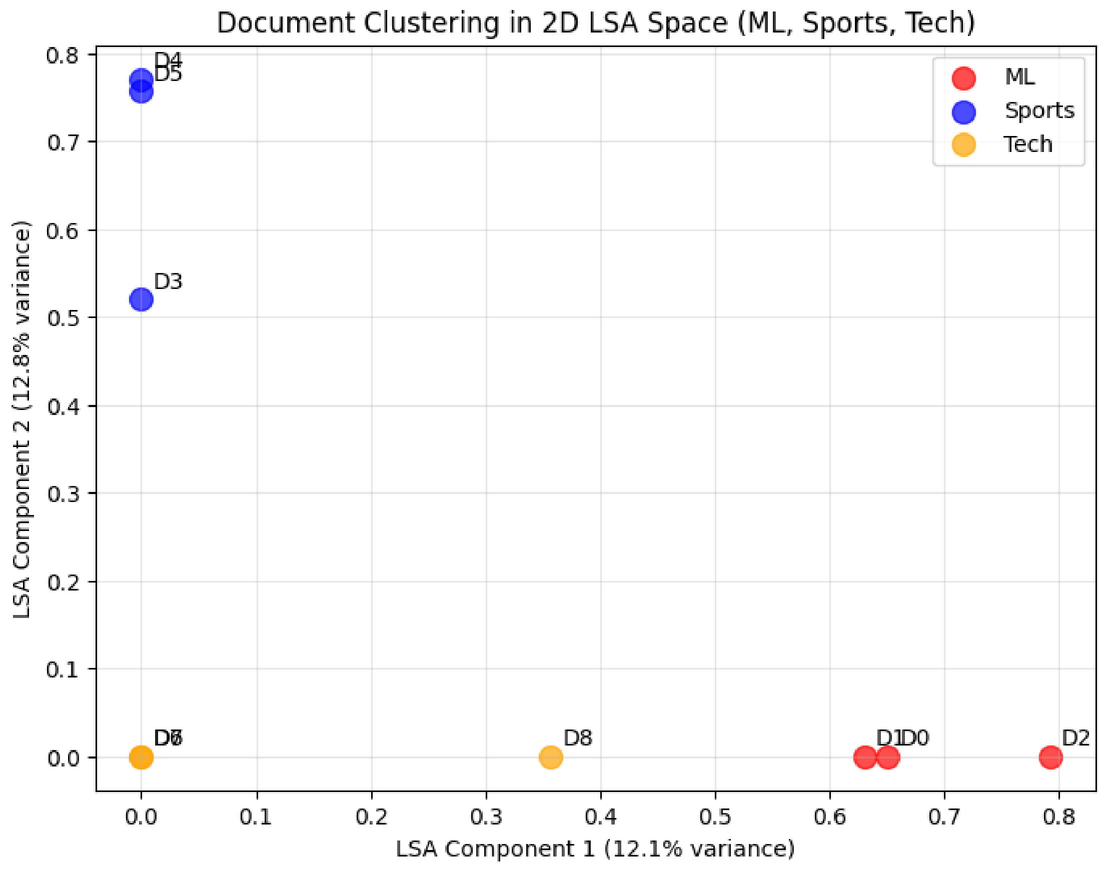


- This low-rank approximation **smooths noise** and reveals **latent semantic dimensions**.

Latent Semantic Analysis (LSA)

- Corpus (3 semantic clusters):
  - Doc 0 (ML): *'machine learning with neural networks'*
  - Doc 1 (ML): *'deep learning requires lots of data'*
  - Doc 2 (ML): *'machine learning algorithms train on data'*
  - Doc 3 (Sports): *'football players run on the field'*
  - Doc 4 (Sports): *'teams of football score many points'*
  - Doc 5 (Sports): *'teams need a football manager on the pitch'*
  - Doc 6 (Tech): *'computers process information quickly'*
  - Doc 7 (Tech): *'programming languages create software'*
  - Doc 8 (Tech): *'databases store structured data'*

- Building TF-IDF Matrix
  - Term (Vocab) size: 31
  - TF-IDF matrix shape: (9, 31)
    - *(No. of docs, No. of term dimensions)*
  - LSA matrix shape: (9, 2)
    - *(No. of docs, No. of latent dimensions)*



Today

IR – Introduction, Evaluation

1 Introduction

- Inverted Index
- Search & Relevance
- TF-IDF & BM25
- LSA

2 **Evaluation**

- Precision & Recall
- MRR & MAP
- nDCG

Evaluation

- We evaluate systems to observe concrete evidence for a hypothesis
  - Is our system better than the other one?

- IR systems are hard to evaluate
  - Ambiguity – what is relevant? In which context? Humans differ a lot …
  - Collection size – explosion of query-document pairs

- Different types of result quality evaluation:
  - **Intrinsic**: Fixed set: same collection, query set & labels
  - **Extrinsic**: Observe behavior of users (in production system)*

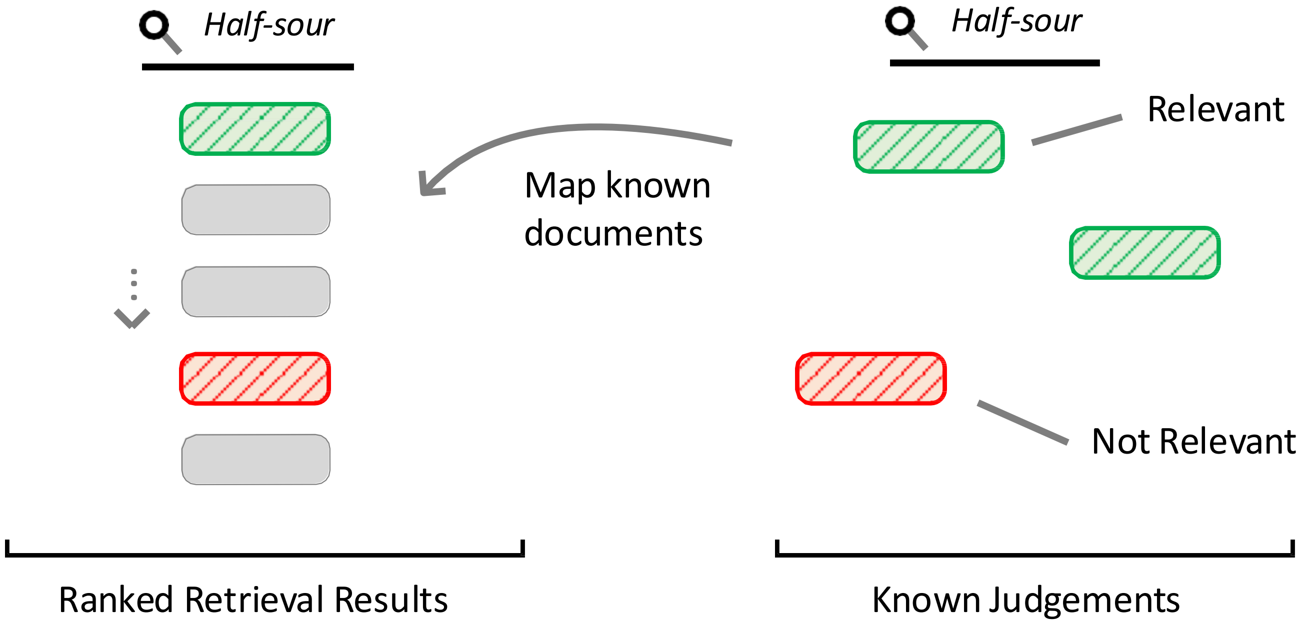
* Could also be a user study, beta version, etc…

The World of Evaluation

- Today we focus on evaluating the result quality of our own IR system
  - Does a document contain the answer for our query?
- Many other possibilities:
  - Efficiency
    - How fast can we index, return results for a query, how large becomes our index on disk?
  - Fairness, diversity, content quality, source credibility, effort, …
  - Retrieval in the context of a larger goal
    - How many products, services do we sell through search
    - How well does our website integrate with Google, Bing, etc.. (SEO)
      - Optimizing a Blackbox

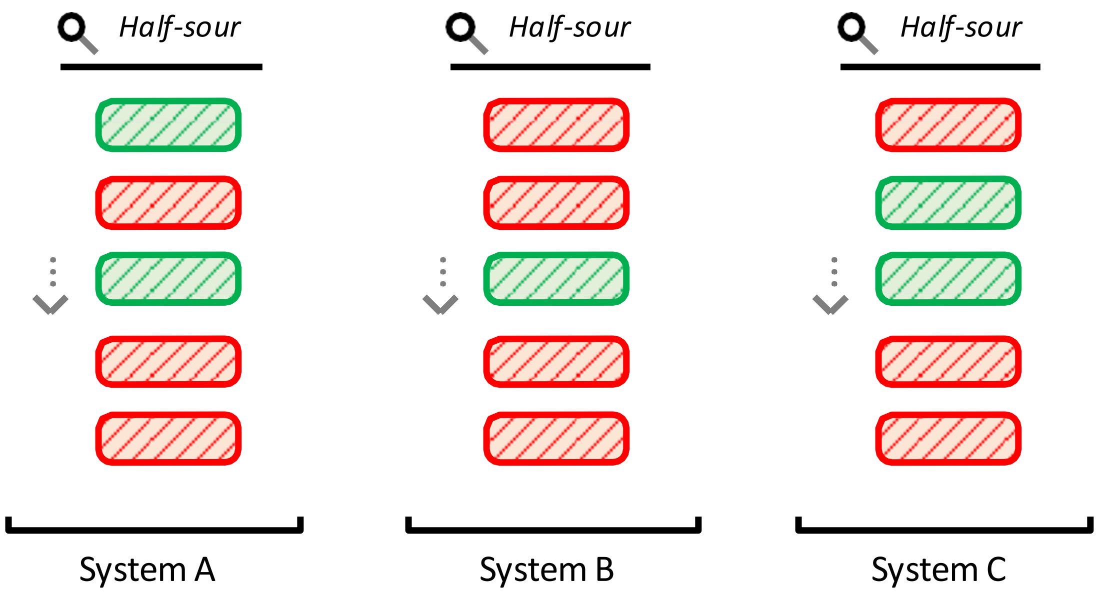
Extrinsic Evaluation Setup

- Quality of systems, that produce ranked list of documents
- Compared by a pool of judgements (does not necessarily cover the whole list)
  - Missing judgements are often considered as non-relevant



Comparing Systems

- We have multiple IR systems running on the same documents & same query
- How to compare them? Evaluation metrics to the rescue!



Today

IR – Introduction, Evaluation

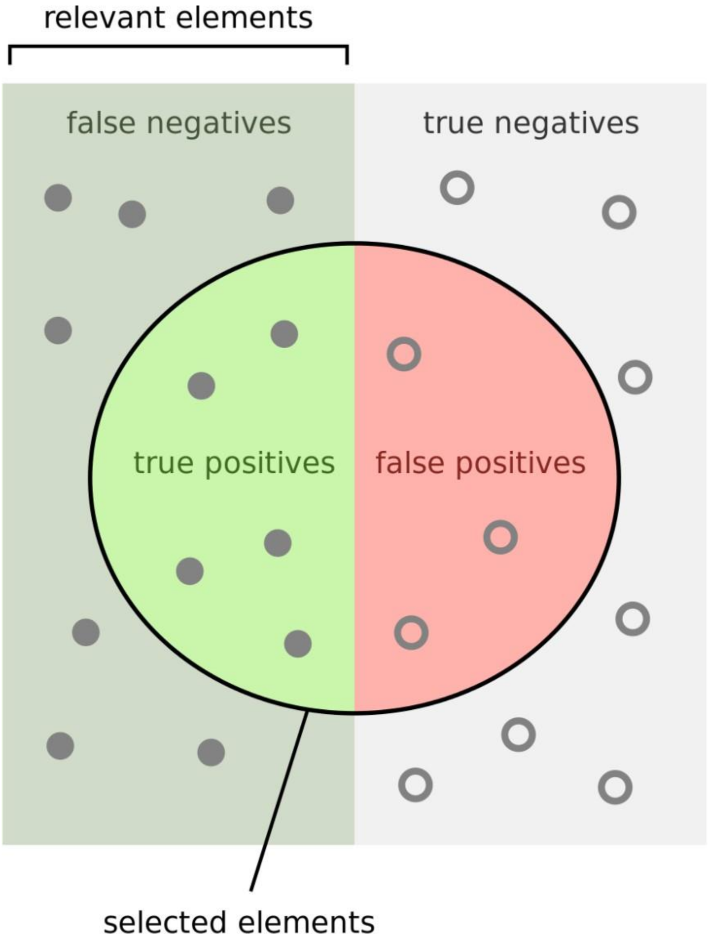
1 **Introduction**

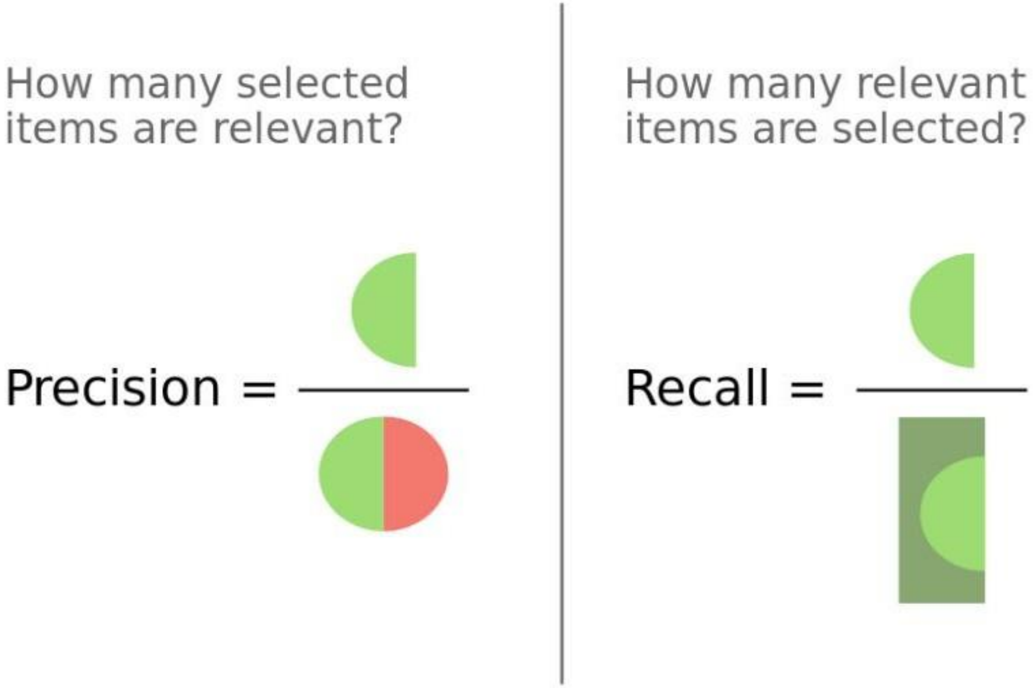
- Inverted Index
- Search & Relevance
- TF-IDF & BM25

2 **Evaluation**

- **Precision & Recall**
- MRR & MAP
- nDCG

Precision & Recall





From: Wikipedia https://en.wikipedia.org/wiki/Precision_and_recall

Precision/recall tradeoff

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

- Combined measure **F-score**:
- allows us to trade off precision against recall
- Mostly used measure: F1 or the harmonic mean of P and R

$$F_1 = 2 \times \frac{P \times R}{P + R}$$

Example for precision, recall, F1

	Relevant	Non-relevant	
Retrieved	20	40	60
Not retrieved	60	1,000,000	1,000,060
	80	1,000,040	1,000,120

$$P = \frac{20}{(20 + 40)} = \frac{1}{3}$$

$$R = \frac{20}{(20 + 60)} = \frac{1}{4}$$

$$F1 = 2 \times \frac{\frac{1}{3} \times \frac{1}{4}}{\frac{1}{3} + \frac{1}{4}} = \frac{2}{7}$$

	Relevant	Non-relevant
Retrieved	TP	FP
Not retrieved	FN	TN

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{(TP + FN)}$$

$$F_1 = 2 \times \frac{P \times R}{P + R}$$

Today

IR – Introduction, Evaluation

1 Introduction
- Inverted Index
- Search & Relevance
- TF-IDF & BM25

2 Evaluation
- Precision & Recall
- **MRR & MAP**
- nDCG

Ranking List Evaluation Metrics

- Binary labels
  - **MRR:** Mean Reciprocal Rank
  - **MAP**: Mean Average Precision
- Graded labels
  - **nDCG**: normalized Discounted Cumulative Gain

- Typically we measure at a cutoff @k of the top retrieved documents
  - MAP, Recall: @100, @1000
  - Precision, MRR, nDCG: @5, @10, @20

Some nice explanations: https://medium.com/swlh/rank-aware-recsys-evaluation-metrics-5191bba16832

MRR: Mean Reciprocal Rank

Users look at results from the top; gets annoyed pretty fast; stops once they found the first relevant; doesn't care about the rest

Mean over all queries

$$MRR(Q) = \frac{1}{|Q|} * \sum_{q \in Q} \frac{1}{FirstRank(q)}$$

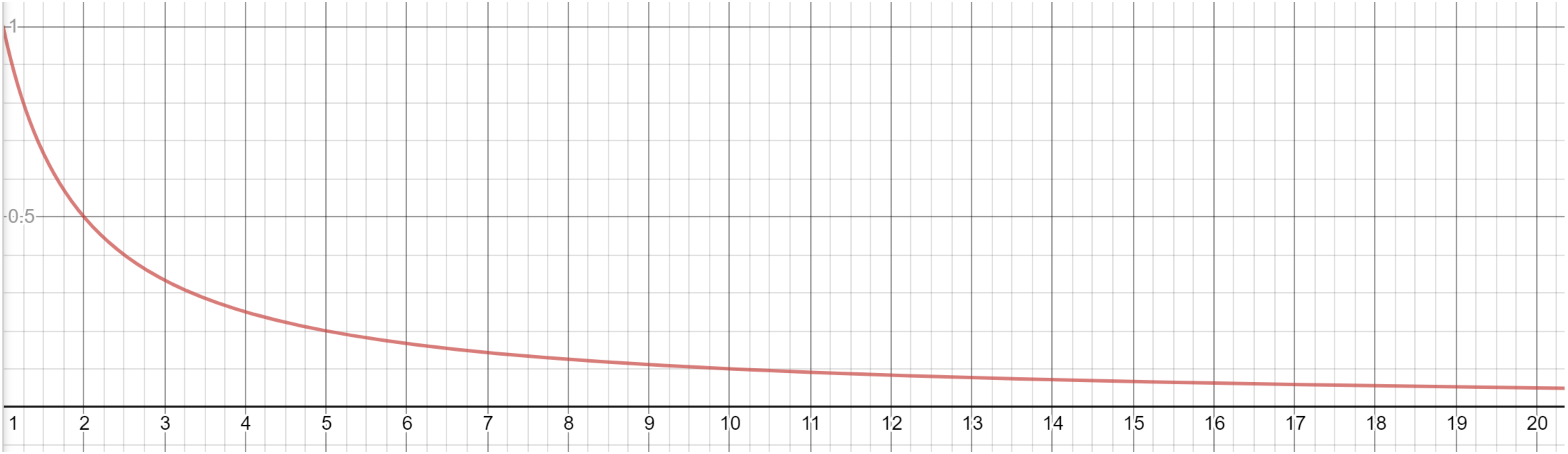
Reciprocal Rank

- MRR puts the focus on the first relevant document
- Applicable with sparse judgements or assuming users are satisfied with one relevant document

Q	$\|Q\|$	$FirstRank(q)$
Query Set	Number of Queries	Returns the Rank of the first relevant document for 1 query

MRR: The Reciprocal Rank

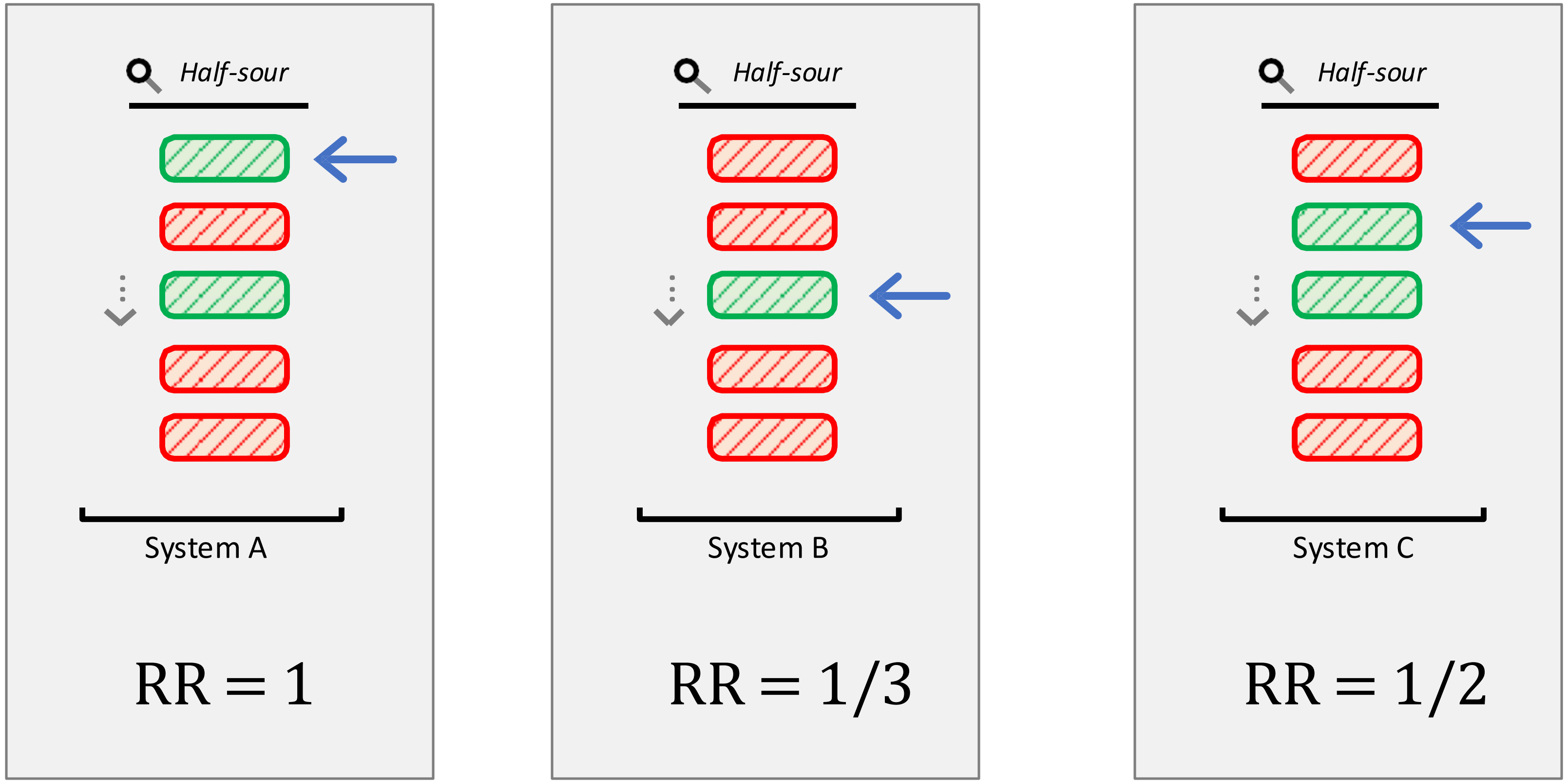
- Reciprocal Rank: $\frac{1}{x}$
- Very strongly emphasis the first position



* x is plotted continuously, but in MRR x is discrete with the position in step size of 1

MRR: An Example

- Example for Reciprocal Rank:



MAP: Mean Ave[redacted]Precision

Users look at results [redacted] every time they [redacted] relevant document, they look [redacted] full picture of what has been before

Mean over all queries

Precision per relevant doc

$$MAP(Q) = \underbrace{\frac{1}{|Q|}} * \sum_{q \in Q} \frac{\sum_{i=1}^{k} P(q)_{@i} * r\,\text{[redacted]}}{|rel(q)|}$$

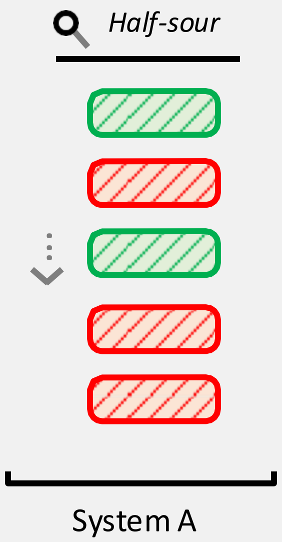
Average Precision

- MAP squeezes complex evaluation [redacted]
- MAP corresponds to the area under the *Precision-Recall curve*

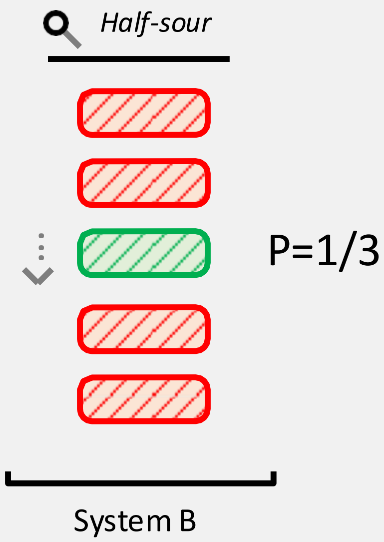
Q	$\|Q\|$	$P(q)_{@}$	$rel(q)$	$\|rel(q)\|$
Query Set	Number of Queries	Precision of query q after first i documents	Binary Relevance of doc at position i	Number of relevant documents

MAP: Mean Average Precision

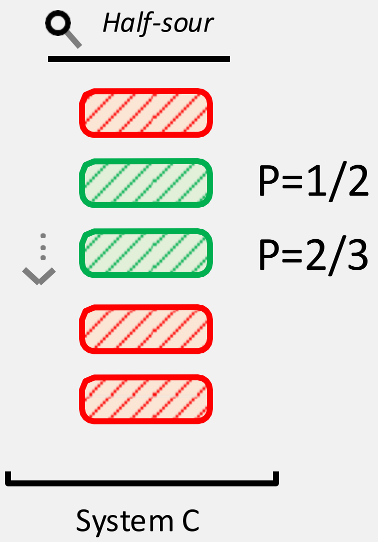
- Example for Average Precision (2 relevant docs)
  - Mean is then calculated for multiple queries, for each system



$$AP = \frac{1+\frac{2}{3}}{2} = 0.83$$



$$AP = \frac{\frac{1}{3}}{2} = 0.16$$



$$AP = \frac{\frac{1}{2}+\frac{2}{3}}{2} = 0.58$$

Today

IR – Introduction, Evaluation

1 Introduction

- Inverted Index
- Search & Relevance
- TF-IDF & BM25

2 Evaluation

- Precision & Recall
- MRR & MAP
- **nDCG**

Graded Relevance

- Previous metrics all use binary relevance labels
  - Simple enough or too simple?
- Major problem: Of course there can be a difference in importance of relevance
  - Binary labels can not distinguish
- Graded relevance allows to assign different values of relevance
  - Can be floating point or fixed set of classes for manual annotation
  - Fixed set of classes for manual annotation
  - Floating point can be used when relevance inferred from logs

Common Graded TREC Relevance Labels

[3] Perfectly relevant: Document is dedicated to the query, it is worthy of being a top result in a search engine.

[2] Highly relevant: The content of this document provides substantial information on the query.

[1] Relevant: Document provides some information relevant to the query, which may be minimal.

[0] Irrelevant: Document does not provide any useful information about the query

nDCG: normalized Discounted Cumulative Gain

Users take for each document the relevance grade and position into account, normalize by best possible ranking per query

$$DCG(q) = \sum_{d \in D, i=1} \frac{rel(d)}{log_2(i+1)}$$

$$nDCG(Q) = \frac{1}{|Q|} * \sum_{q \in Q} \frac{DCG(q)}{DCG(sorted(rel(q)))}$$

- nDCG compares actual results with maximum per query
- Relevance is graded
- nDCG@10 most commonly used in modern offline web search evaluation

Q	$\|Q\|$	D	$rel(d)$	$rel(q)$	$sorted()$
Query Set	# of Queries	Single Doc. Result list	Relevance grade for single query-doc pair	List of all relevance grades for a query	Return graded documents by descending relevance

nDCG: A Closer Look

Discounted cumulative gain

$$DCG(q) = \sum_{d \in D,\, i=1} \frac{rel(d)}{log_2(i+1)}$$

Gain (relevance value, commonly 0 -> 3)

Position Discounting

$$nDCG(Q) = \frac{1}{|Q|} * \sum_{q \in Q} \frac{DCG(q)}{DCG(sorted(rel(q)))}$$

Actual Results

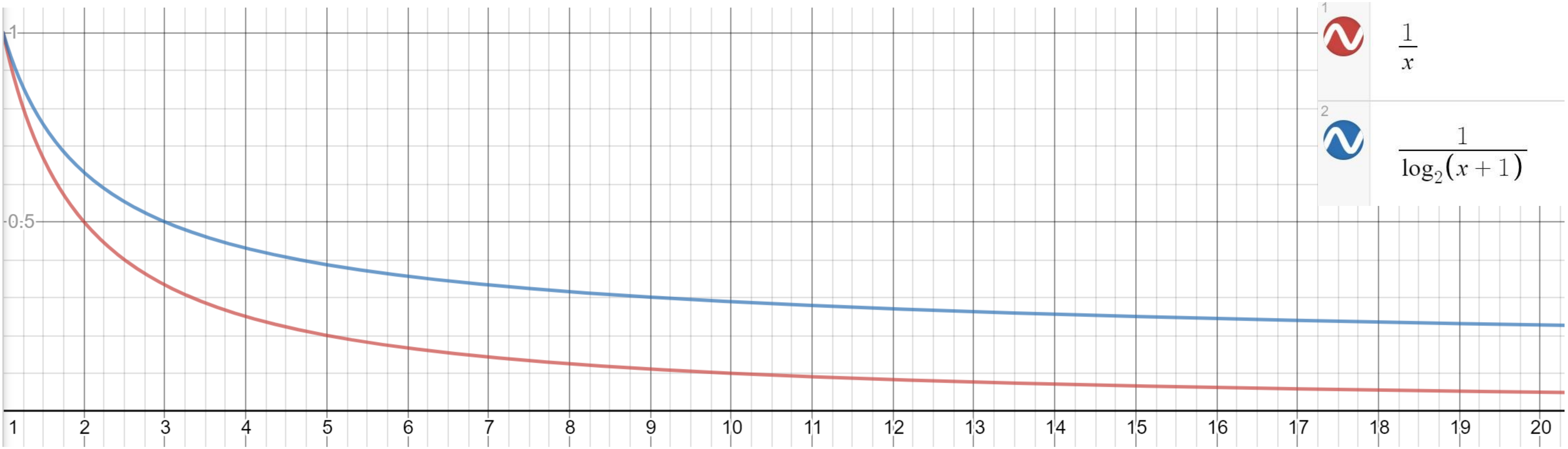
Best possible sorting (ground truth)

Mean over all queries

Q	$\lvert Q \rvert$	D	$rel(d)$	$rel(q)$	$sorted()$
Query Set	# of Queries	Single Doc. Result list	Relevance grade for single query-doc pair	List of all relevance grades for a query	Return graded documents by descending relevance

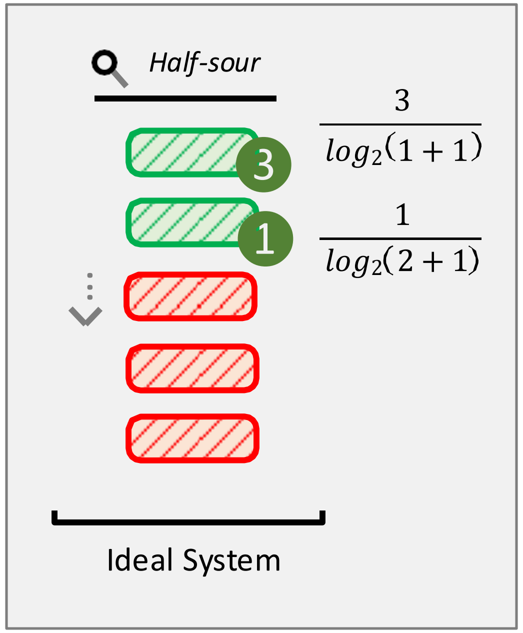
nDCG: Position Discounting

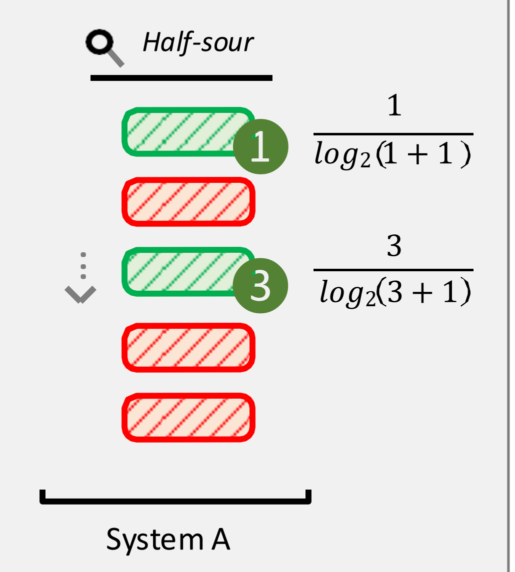
- Comparing the document position discount with reciprocal rank
  - Only for binary case rel=1
- nDCG discounts less than MRR



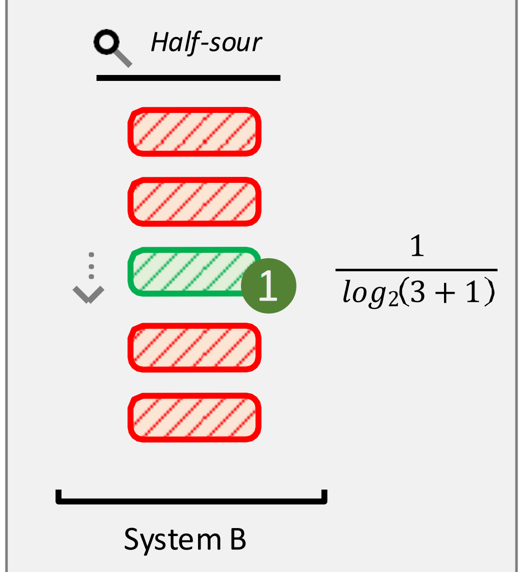
nDCG: Example

- Assuming two differently relevant docs (rel = 3 & 1)
  - Ideal DCG = $\frac{3}{log_2(1+1)} + \frac{1}{log_2(2+1)} = 3.63$

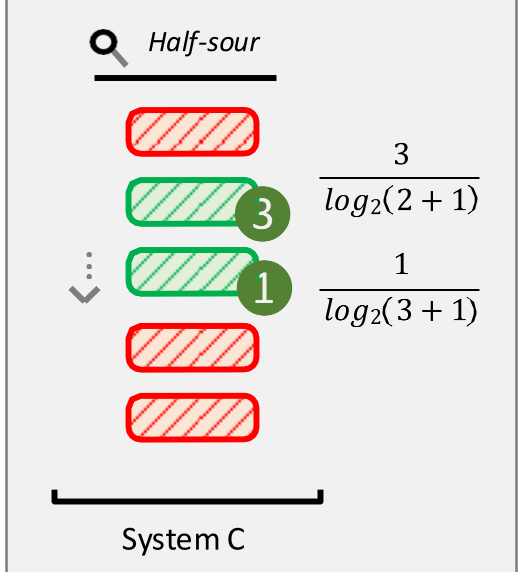




$nDCG = \frac{1+1.5}{3.63} = 0.69$



$nDCG = \frac{0.5}{3.63} = 0.14$



$nDCG = \frac{1.9+0.5}{3.63} = 0.66$

Summary: Part 2

1. We compare systems with a set of query and document relevance labels

2. Binary metrics (MRR & MAP) are a solid foundation for evaluation

3. Graded relevance allows for more fine-grained metrics (nDCG)