Relevance & Scoring

Sebastian Hofstätter

sebastian.hofstaetter@tuwien.ac.at

TU WIEN

Today

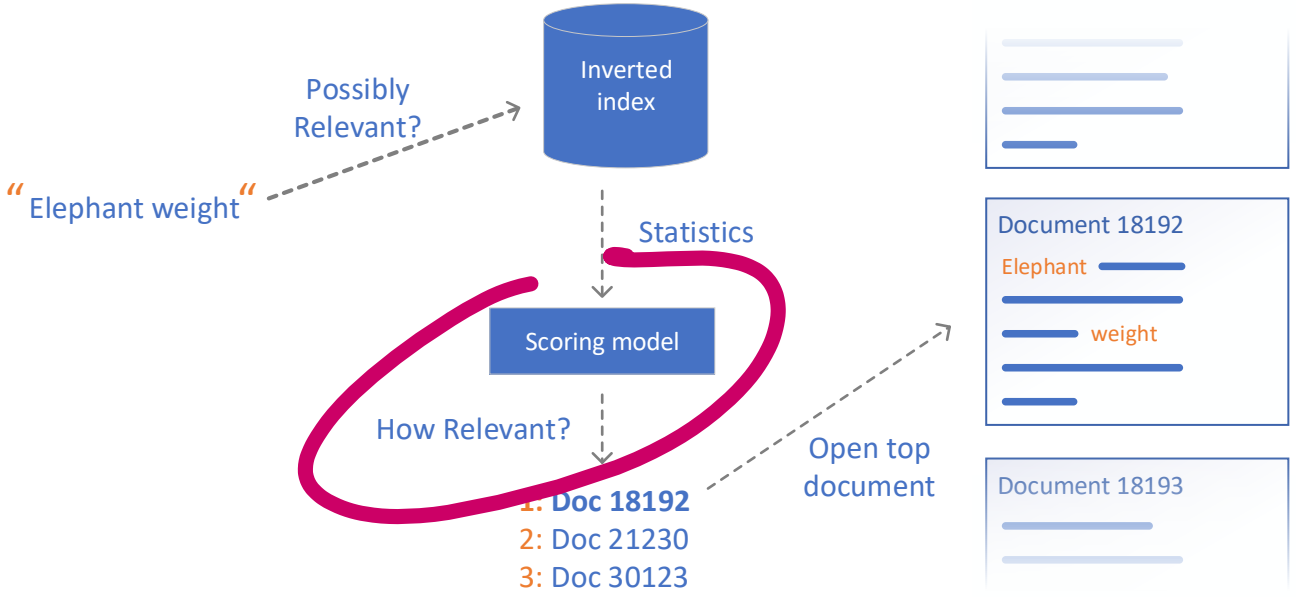
With a break after VSM

Relevance & Scoring

1. Relevance
2. TF-IDF
3. Vector Space Model
4. BM25

Some materials taken from: *Introduction to IR by Manning lecture materials*
Mihai Lupu's & Navid Rekabsaz' previous lecture slides

Recap - Inverted Index query workflow



- Inverted Index stores statistics about terms & documents
- Scoring model works with those statistics
- Sort documents based on relevance score – retrieve most relevant documents

Relevance

+ Some notes on the search workflow

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

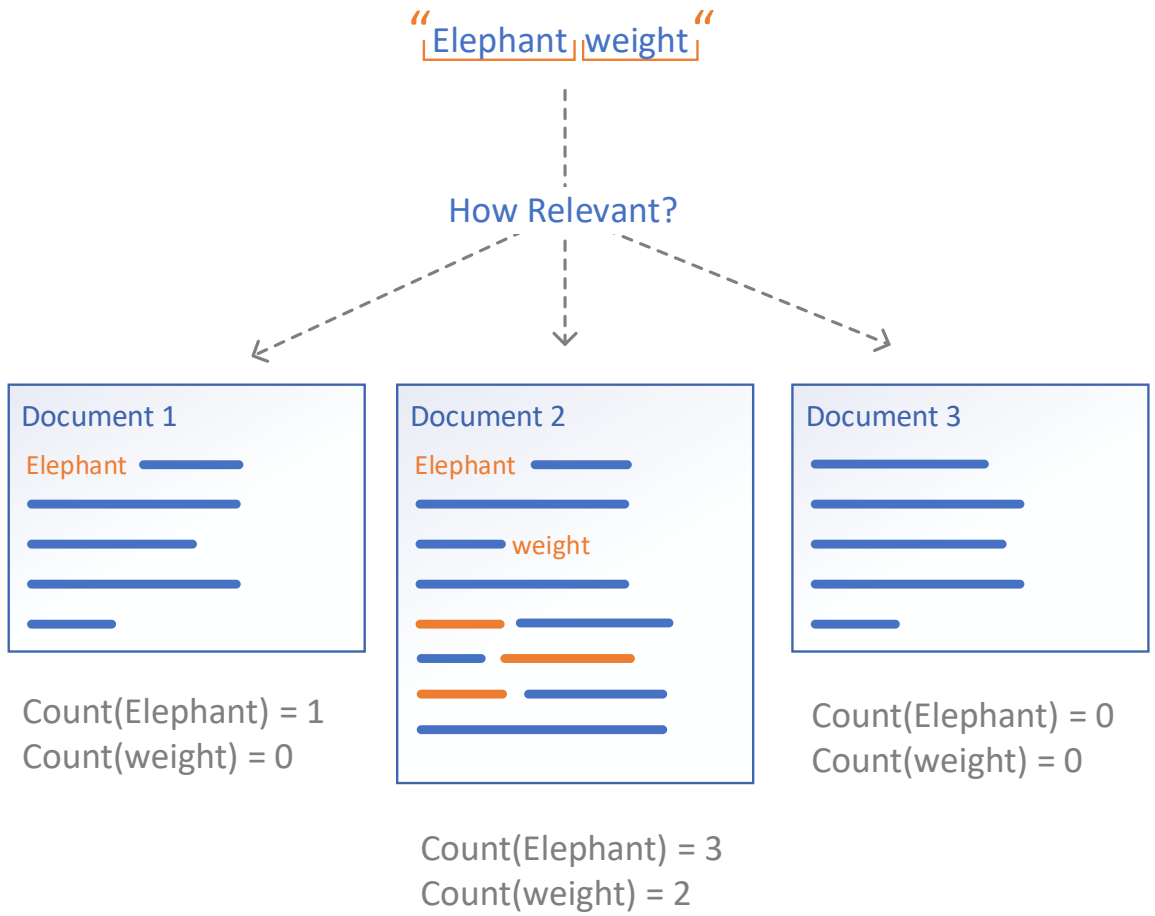
if *d* not in *Scores* **do** *Scores*[*d*]=0

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

- Words are meaningless – we see them as discrete symbols
- Documents are therefore a stream of meaningless symbols

- We try to find patterns or trends

- Understanding of relevance probably requires deep understanding of language and/or the human brain
  - A step in this direction → using neural networks for relevance computation

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

TF-IDF

Term Frequency – Inverse Document Frequency

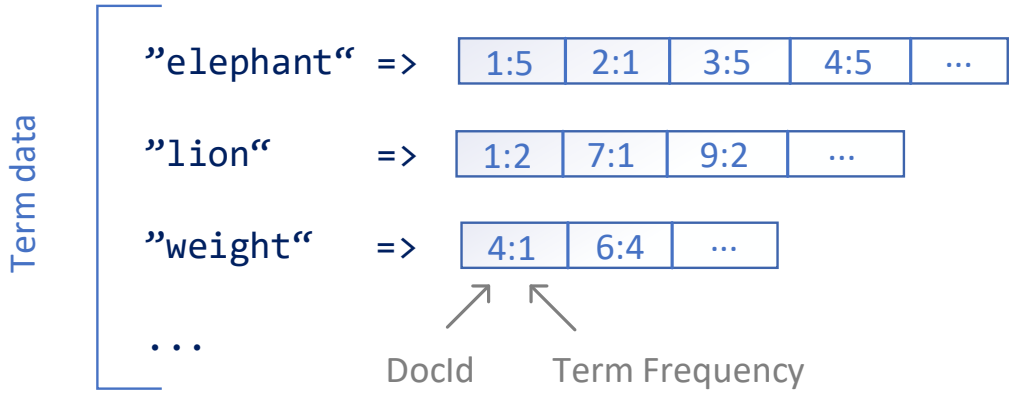
Term Frequency – conceptional data view

- **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

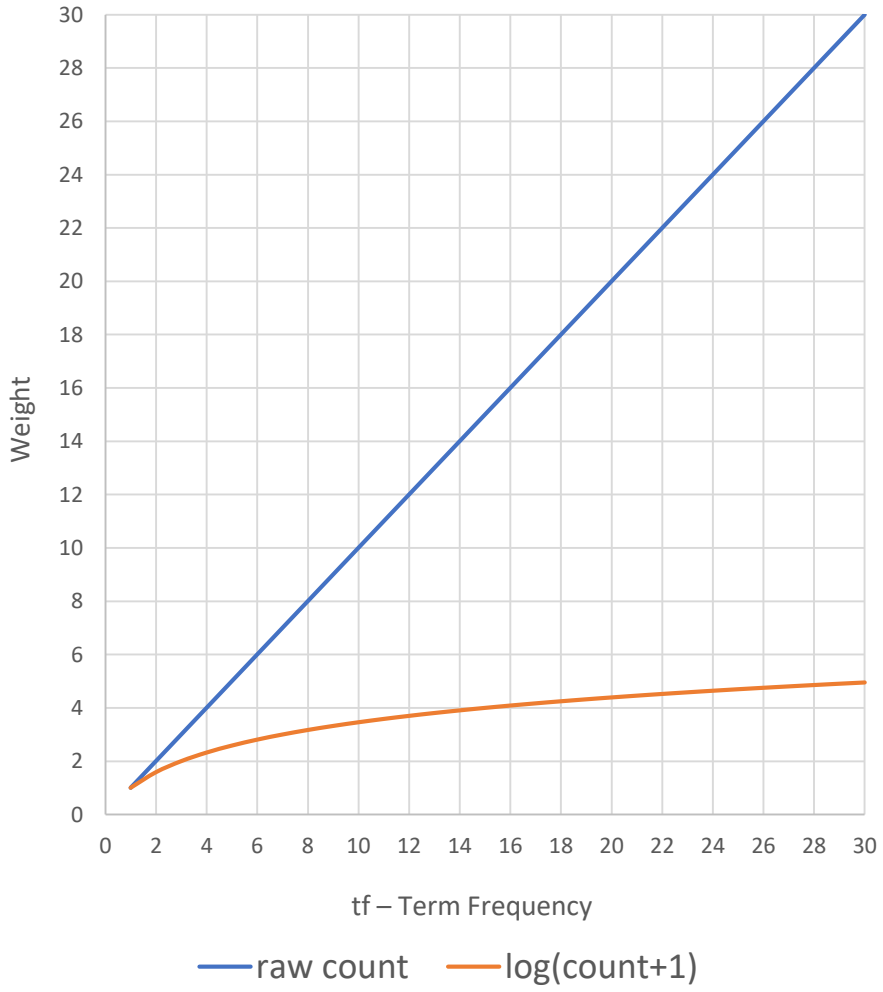
- Inverted index saves only non-0 entries, not the whole matrix
  - Otherwise we would waste a lot of storage capacity
- Therefore not good at random lookups into the document column
  - Needs to iterate through the posting list to find the correct document
  - However, for scoring models $tf_{t,d}$ 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 stop words
- Consider a term in the query that is rare in the collection
  - e.g., *TUWIEN in a news corpora*
- A document containing this term is very likely to be relevant to the query *TUWIEN*

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

IDF – Inverse Document Frequency

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

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

- 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.

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

IDF – Inverse Document Frequency

- Does *idf* have an effect on ranking for one-term queries, like query "iPhone"?

- *idf* has no effect on ranking one term queries
  - *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

TF-IDF

$$TF\_IDF(q,d) = \sum_{t \in T_d \cap T_q} log(1 + tf_{t,d}) * log(\frac{|D|}{df_t})$$

increases with the number of occurrences within a document

increases with the rarity of the term in the collection

- 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

$\sum_{t \in T_d \cap T_q}$ Sum over all query terms, that are in the index

$tf_{t,d}$ Term frequency

$|D|$ Total # of documents

df_t # of Documents with $tf_{t,d} > 0$

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: Latent Semantic Analysis (LSA) works better with tf-idf weights

VSM

Vector Space Model

Terms & Docs – conceptional data view

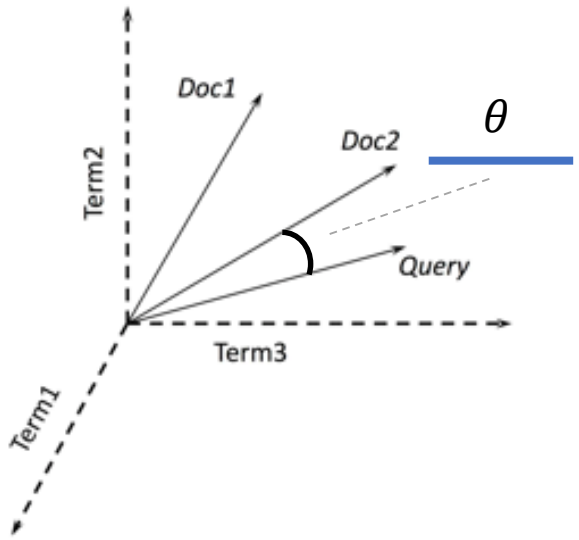
- $tf_{t,d}$ number of occurrences of term *t* in document *d*
- Why don't we interpret the rows & columns as vectors?
- We can compute a score with those vectors!

		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

VSM - Vector Space Model

- Documents and queries are represented as vectors in a t-dimensional space
  - T is the collection of distinct terms in collection with M members
  - N is the number of documents in collection
  - M≈[100K to 500K] N≈[100 to many millions]
- Document vector $\vec{d} = (w_{t_1,d}, w_{t_2,d}, \dots, w_{t_M,d})$
  - $w_{t,d}$ is referred to as term weighting
- Query vector $\vec{q} = (q_{t_1}, q_{t_2}, \dots, q_{t_M})$

Cosine similarity



$$\text{sim}(d, q) = \cos(\theta)$$

$$= \frac{\vec{d} \cdot \vec{q}}{|\vec{d}||\vec{q}|}$$

- Cosine similarity measures direction of vectors, but not the magnitude
- Not a distance – but equivalent to Euclidean distance of unit (length=1) vectors

θ	Angle between two vectors
$\vec{q}$	Vector of query q
$\vec{d}$	Vector of doc d
$\vec{d} \cdot \vec{q}$	Dot product

$$= \sum_{i=1}^{dim} d_i * q_i$$

VSM with cosine similarity

$$VSM(d,q) = \cos\left(\vec{d},\vec{q}\right) = \frac{\vec{d}\cdot\vec{q}}{|\vec{d}||\vec{q}|} = \frac{\sum_{t\in T} w_{t,d} * q_t}{\sqrt{\sum_{t\in T} {w_{t,d}}^2 * \sum_{t\in T} {q_t}^2}}$$

- A query or document is projected as a vector where terms represent the dimensions
- Scoring function is the cosine similarity between the vectors
- Intuition for relevance: documents with a similar direction as the query are more likely relevant

$\vec{q}$	Vector of query q
$\vec{d}$	Vector of doc d
$\vec{d}\cdot\vec{q}$	Dot product
$w_{t,d}$	Entry in the doc vector at position of term t
q_t	Entry in the query vector at position of term t

VSM – Using tf-idf

$$VSM_{tf\_idf}(d,q) = \sum_{t \in T_d \cap T_q} \frac{\log(1 + tf_{t,d}) * \log({}^{|D|}/_{df_t})}{\sqrt{\sum_{t \in T_d} \left[\log(1 + tf_{t,d}) * \log({}^{|D|}/_{df_t})\right]^2}}$$

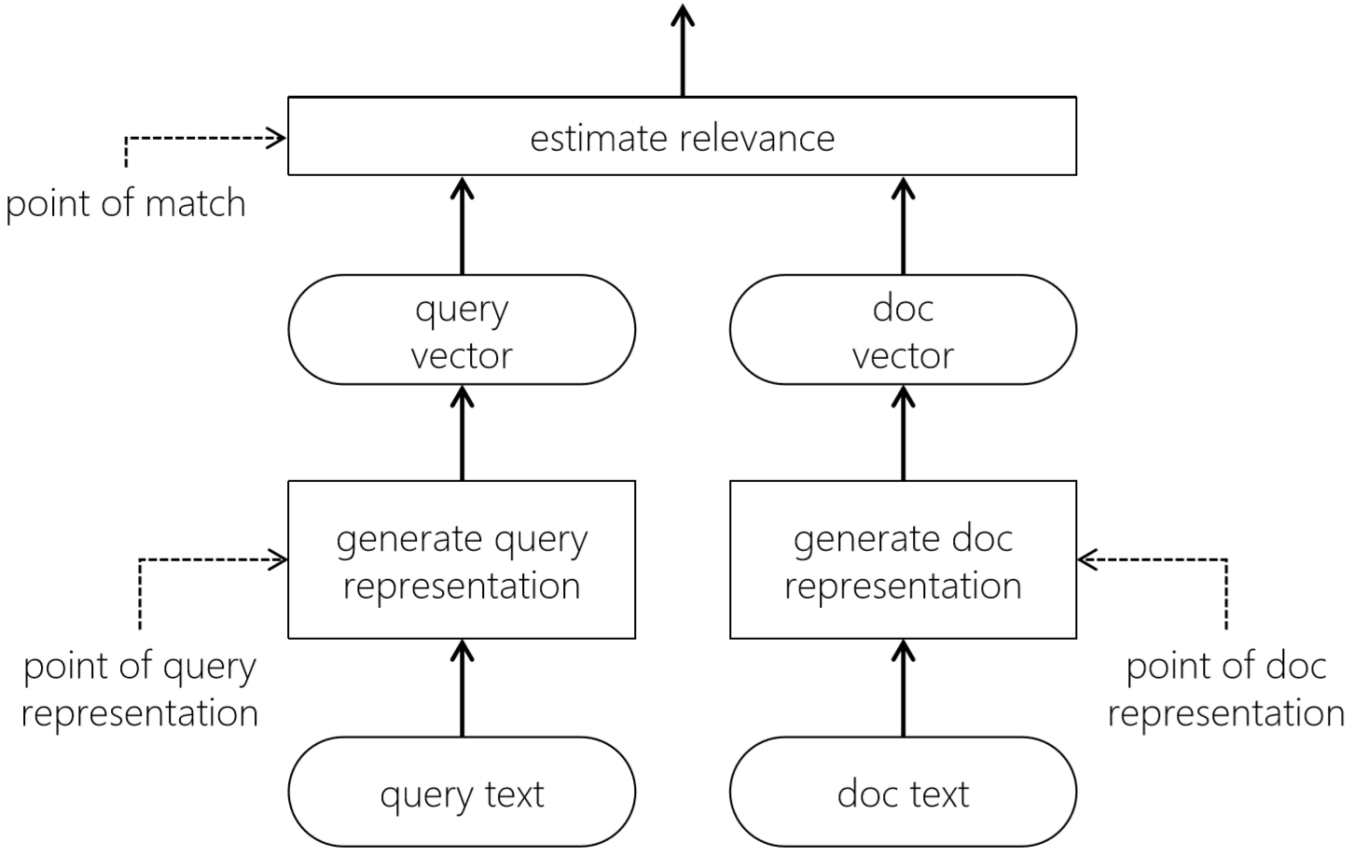
- As a common practice, VSM model uses *tf-idf* as the weight
- The formula incorporates the normalization of the cosine similarity
- It is calculated **only** on the common terms between document and query (we don't use 0-entries from our conceptional data view)

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

Break 

Do you have questions in the meantime?

Research Example: Neural Networks for IR



- "Learning to Rank" instead of handcrafted formulas
- Moving from BOW counting to vector representations
- Needs a lot of data
- More complex than "simple" classification tasks

Mitra and Craswell, "An Introduction to Neural Information Retrieval" Foundations and Trends® in IR, 2018
https://www.microsoft.com/en-us/research/publication/introduction-neural-information-retrieval/

BM25

“BestMatch25”

Probabilistic retrieval

- Retrieval is inherently uncertain
- Probabilities provide a foundation for uncertain reasoning
- The probabilistic model seeks to ground retrieval in answering, for each document and each query, the “Basic Question”:

  What is the probability that *this* document is relevant to *this* query?

Probabilistic retrieval

- Strong (and pretty unrealistic) independence assumptions:
  - Terms in a document are independent
  - Documents are independent

- With those assumptions:
  - Theoretical foundation for the search workflow

$$P(rel|d,q) \propto_q \frac{P(rel|d,q)}{P(\overline{rel}|d,q)} \quad \dots \quad \prod_{i \in q} \frac{P(TF_i = tf_i|rel)}{P(TF_i = tf_i||\overline{rel})} \quad \dots \quad \underbrace{\sum_{q,tf_i>0} w_i}_{=score(q,d)}$$

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

Aboutness / eliteness terms

- Hidden property for document – query term pair: eliteness
  - If the term is elite in the document, the document is about the concept denoted by the term
  - Eliteness is what connects relevance and term frequency
  - Eliteness is binary
- A document is built up from elite & non-elite terms
- We assume for each elite & non-elite terms there is a Poisson distribution (2 Poisson model)
  - Does not account for different document lengths
  - Exhibits interesting eliteness <-> term frequency saturation property

Term Frequency Saturation

- 2-Poisson model eliteness <-> term frequency properties:

1. Eliteness weight increases monotonically with tf
2. But asymptotically approaches a maximum value as tf → ∞

- Saturation: any one term's contribution to the document score cannot exceed a saturation point (the asymptotic limit)

- Very useful – therefore used in the BM25 model

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)

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

- Assuming we have no additional relevance information
  - If we do: Use RSJ (Robertson/Spärck Jones) weight instead of IDF
- Simpler than the original formula
  - Over time it was shown that more complex parts not needed

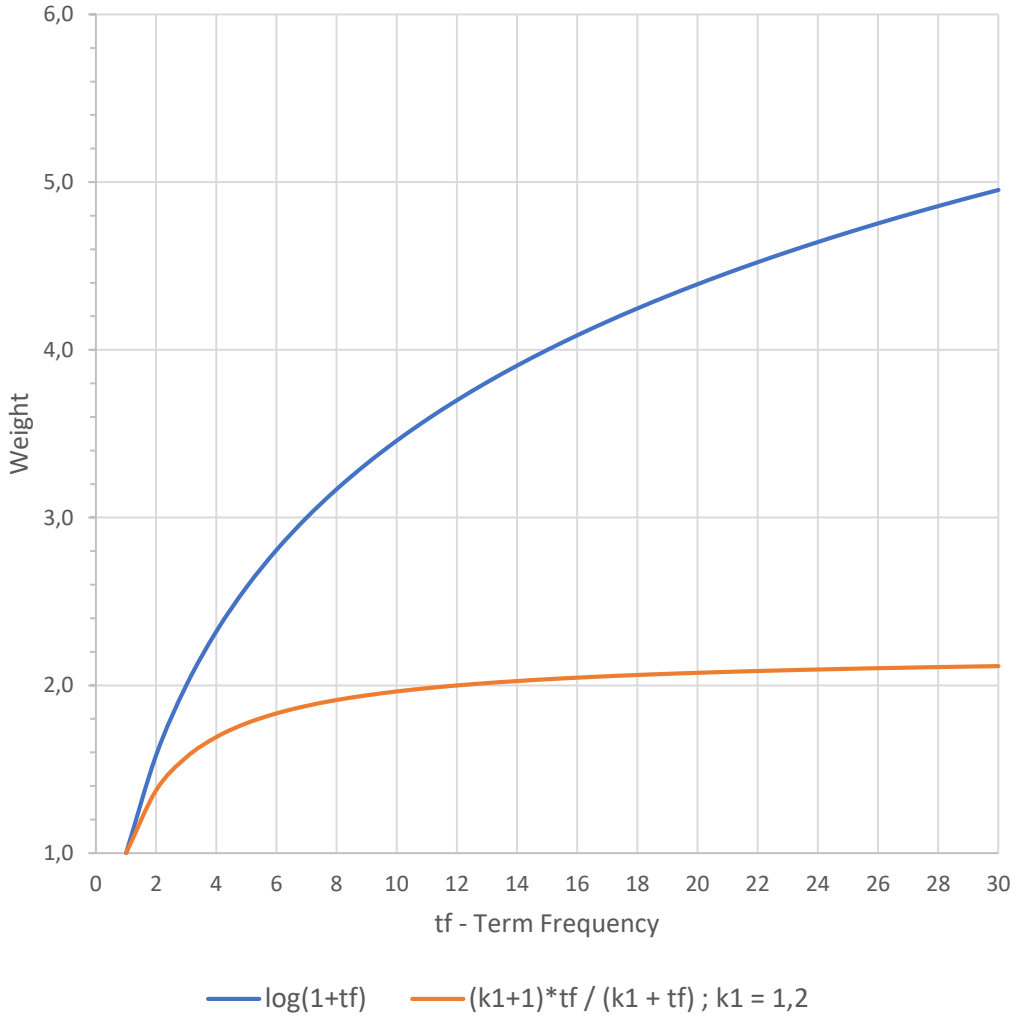
Details (a lot of them): The Probabilistic Relevance Framework: BM25 and Beyond
http://www.staff.city.ac.uk/~sb317/papers/foundations_bm25_review.pdf

$\sum_{t \in T_d \cap T_q}$	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 $tf_{t,d} > 0$
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: $\log(\text{tf}) * \log(|\text{D}|/\text{df})$
  - doc1: $11 * 7 + 1 * 10 = \mathbf{87}$
  - doc2: $5 * 7 + 4 * 10 = \mathbf{75}$
- BM25: $k_1 = 2$
  - doc1: $7 * 3 + 10 * 1 = \mathbf{31}$
  - doc2: $7 * 2.67 + 10 * 2.4 = \mathbf{42.7}$

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

BM25F

- BM25 only covers the document as 1 unstructured heap of words
- Real world use case: documents have at least some structure
  - Title, abstract, infobox, headers …
  - Anchor text in web pages describing a page (see Google paper in Lecture 1)

- BM25**F** allows for multiple fields (or “streams”) in a document
  - For example - 3 streams per doc: title/abstract/body

- BM25**F** allows to assign different weights to the individual streams

BM25F (as defined by Robertson et al. 2009)

$$BM25F(q,d) = \sum_{t \in T_d \cap T_q} \frac{\widetilde{tf}_{t,d}}{k_1 + \widetilde{tf}_{t,d}} * log \frac{|D| - df_t + 0.5}{df_t + 0.5}$$

$$\widetilde{tf}_{t,d} = \sum_{s=1}^{S_d} w_s \frac{tf_{t,s}}{(1 - b_s) + b_s \frac{sl_s}{avgsl})}$$

- Assuming we have no additional relevance information – if we do use RSJ
- Shared IDF might be problematic, could be improved

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
$\sum_{s=1}^{S_d}$	Sum over streams for one doc
$tf_{t,s}$	Term frequency in the stream s
sl_s	Stream length
$avgsl$	Average stream length in index
$\|D\|$	Total # of documents
df_t	# of Documents with $tf_{t,d} > 0$
w_s	Stream weight
k_1, b_s	Hyperparameters

BM25F

- BM25F first combines streams and then terms
  - This is different than chaining together BM25 results on different streams
  - The saturation function is applied at the stream level
- This follows the property that the f.e. the title and body of 1 document are not independent from each other
  - And may not be treated as independent
- Naturally, one assigns a higher stream weight to titles and abstracts
  - Exact values have to be found again with evaluating different settings and relevance judgements

Details coming up in the evaluation lecture

Summary: Scoring & Relevance

1. Scoring models capture relevance in a mathematical model

2. TF-IDF uses term and document frequencies to score a query & doc

3. BM25 is a good scoring model grounded in probabilistic retrieval

1. Scoring models capture relevance in a mathematical model
2. TF-IDF uses term and document frequencies to score a query & doc
3. BM25 is a good scoring model grounded in probabilistic retrieval

Thank You