Big Data and Data Mining

Ranking

Flavio Bertini

flavio.bertini@unipr.it

Boolean model limits

- 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 knowledge of the collection
  - Also good for applications: applications can easily make use of 1000s of results for further processing
- 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 model limits: example

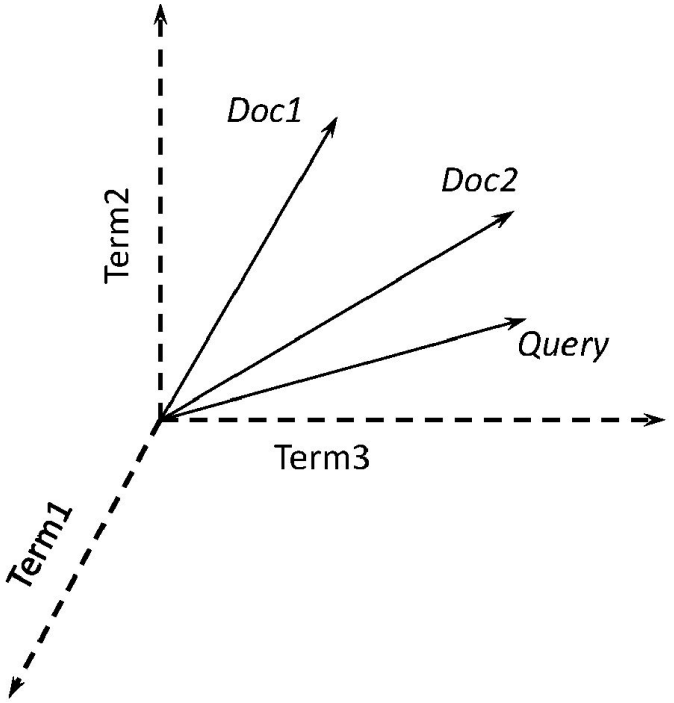
- Boolean queries often result in either **too few** (=0) or **too many** (1000s) results
  - For example, suppose a user is having troubles with his/her network card:
    - Query 1: "*standard user dlink 650*" → 200,000 hits
    - Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes 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

- Rather than a set of documents satisfying a query expression, in **ranked retrieval**, the system returns an ordering over the (**top**) documents in the collection for a query
- When a system produces a ranked result set, large result sets are not an issue
  - We just show the **top *k*** (≈ 10) results
  - We don't overwhelm the user
  - Premise: the ranking model works
- Most known ranking model is Vector Space Model
  - Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. Communications of the ACM

Vector Space Model (VSM)

- Like in the bag-of-word model, documents are **represented by a vector** of "term weights"



- Collection represented by a matrix of **term weights**
  - d_{ij} is the weight of Doc_i for $Term_j$

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \qquad Q = (q_1, q_2, \ldots, q_t)$$

	$Term_1$	$Term_2$	$\ldots$	$Term_t$
Doc_1	d_{11}	d_{12}	$\ldots$	d_{1t}
Doc_2	d_{21}	d_{22}	$\ldots$	d_{2t}
$\vdots$	$\vdots$			
Doc_n	d_{n1}	d_{n2}	$\ldots$	d_{nt}

VSM: vector representation

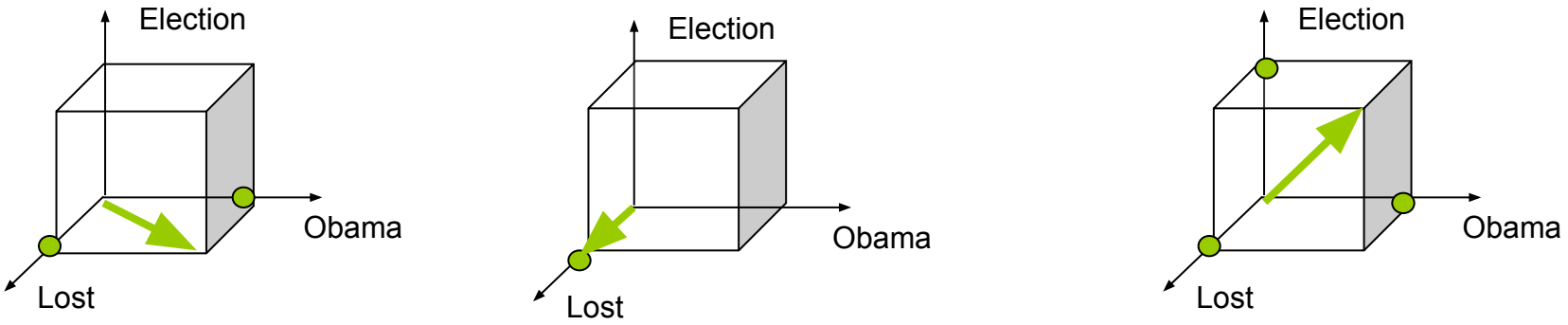
- An example with 3 terms (3-dimensional vector space) and 3 documents

D1 In last year election, Romney **lost election** against **Obama**

D2 I **lost** my faith in Clinton's ability to win

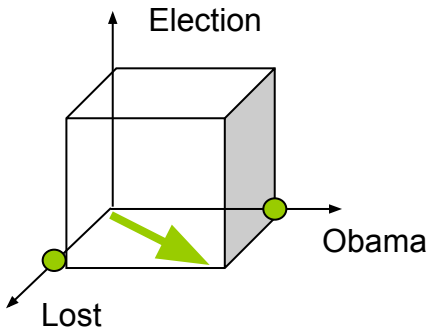
D3 **Obama** has **lost** control on Senate

We consider only the terms in figure



Documents as Vectors

- 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 when you apply this to a web search engine
- These are very sparse vectors - most entries are zero



Queries as Vectors

- **Key idea 1:** Do the same for queries representing them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space
  - proximity = **similarity of vectors**
  - proximity ≈ inverse of distance

- **Reminder**: We do this because we want to get away from the in-or-out Boolean model
- Instead: **rank more relevant documents** higher than less relevant documents

Recap

- In the boolean model, a document is either relevant or non-relevant
- Representing document and queries as vectors, we can compute the **similarity between the vector** of the document and the vector of the query:
  - This is an estimation of the relevance of the document with respect to the query

- We need to define the similarity between vectors but...
  - Before going on we must introduce the **term frequency**

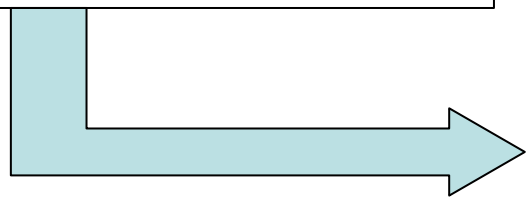
TF: Term Frequencies

- So far, vectors were binary, representing the **presence or absence** of terms in a document
- We would like to assign a **weight** for each term in a vector space:
  - The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*
  - We want to use term frequency (**tf**) when computing query-document match scores

Example: TF as term weight

Three dimensional pictures are useful, but can be misleading for high-dimensional space (more than 3 terms considered):

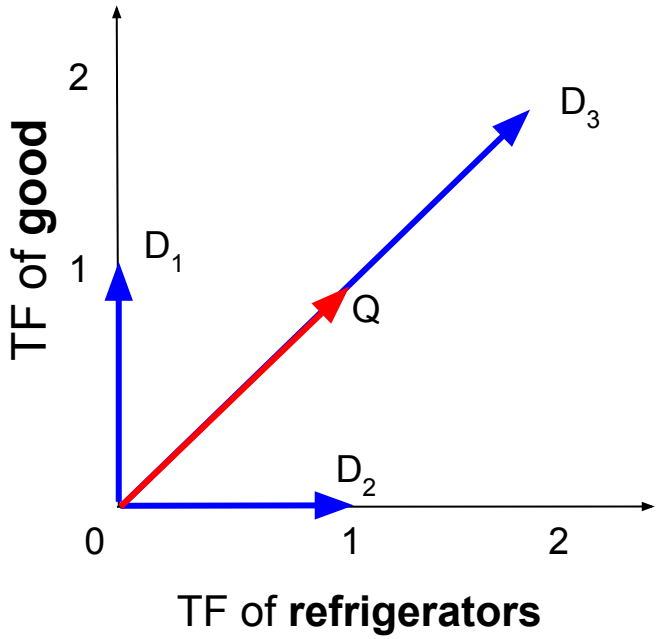
D_1 Tropical Freshwater Aquarium Fish.
D_2 Tropical Fish, Aquarium Care, Tank Setup.
D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.



	Documents			
Terms	**D_1**	**D_2**	**D_3**	**D_4**
Aquarium	1	1	1	1
Bowl	0	0	1	0
Care	0	1	0	0
Fish	1	1	2	1
Freshwater	1	0	0	0
Goldfish	0	0	1	0
Homepage	0	0	0	1
Keep	0	0	1	0
Setup	0	1	0	0
Tank	0	1	0	1
Tropical	1	1	1	2

Vector Space Distance

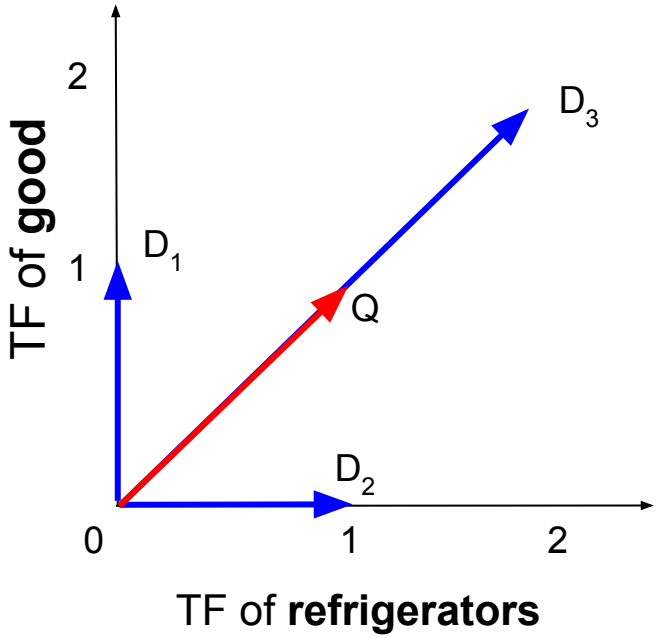
- How to formalize the proximity between two vectors when we have term frequencies?
- Example query Q: “Good refrigerators”
  - D_1: “Good morning to all of you.”
  - D_2: “Don’t put pizza in refrigerators”
  - D_3: “Good Refrigerator Review: top five good refrigerators.”



Note: we are showing only the two interesting dimensions, “good” and “refrigerators”. There is a dimension for each indexed term (*e.g.*, for pizza, morning, review…)

First cut: Euclidean Distance?

- We could consider the straight line distance between the endpoints, *i.e.*, the euclidean distance
  - Distance between Q and D_1 is 1
  - Distance between Q and D_2 is 1
  - Distance between Q and D_3 is $\sqrt{2} \cong 1.414$
- D_3 is the least similar, this doesn't seems right!



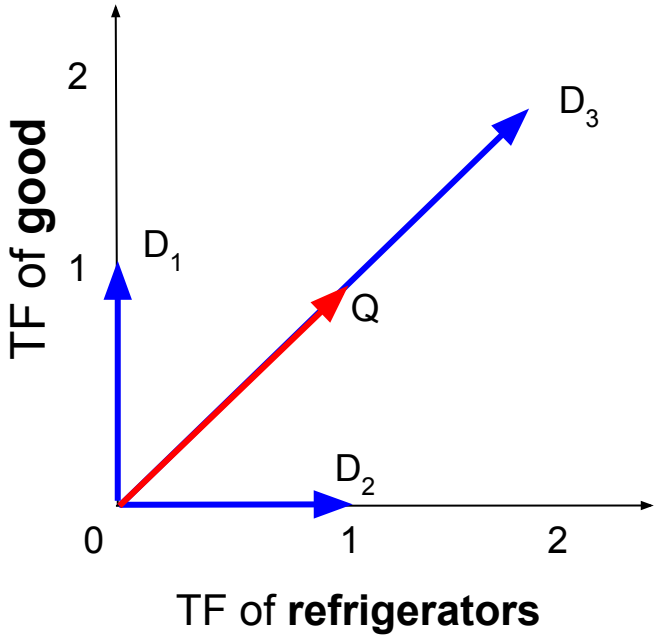
- Remember Q: "Good refrigerators"
  - D_1: "Good morning to all of you."
  - D_2: "Don't put pizza in refrigerators"
  - D_3: "Good refrigerators review: top five good refrigerators."

Angle instead of distance

- Thought experiment: take a document *d* and append it to itself many times. Call this longer 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

Example: Using angles

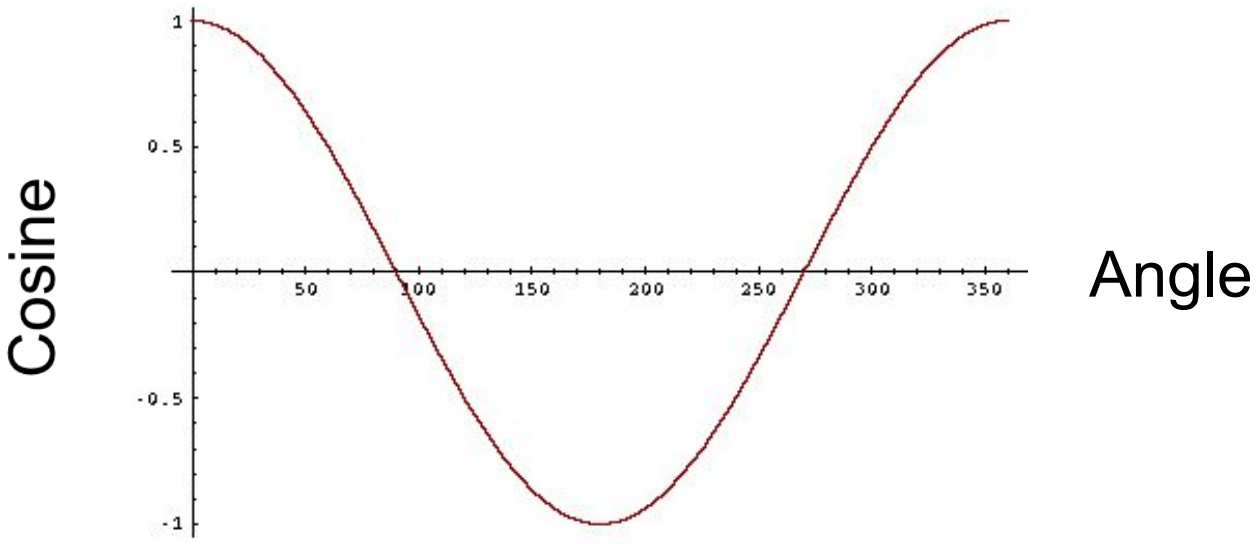
- We now consider the angle between vectors
  - Angle between Q and D_1 is 45°
  - Angle between Q and D_2 is 45°
  - Angle between Q and D_3 is 0°
- D_3 is the most similar, this is what we need!



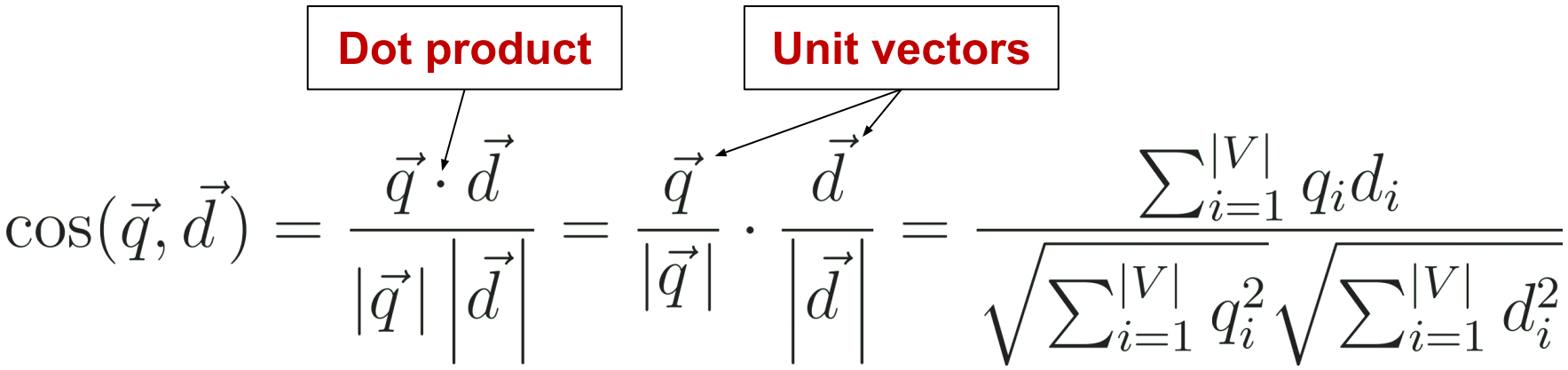
- Remember Q: “Good refrigerators”
  - D_1: “Good morning to all of you.”
  - D_2: “Don’t put pizza in refrigerators”
  - D_3: “Good refrigerators review: top five good refrigerators.”

From Angles to Cosines

- The following two notions are equivalent.
  - Rank documents in increasing order of the angle between query and document
  - Rank documents in decreasing order of *cosine*(query, document)
- Cosine is a monotonically **decreasing** function for the interval [0º, 180º], meaning that it's inversely proportional of the angle



Cosine similarity



- q_i is the tf weight of term *i* in the query
- d_i is the tf weight of term *i* in the document

$\cos(\vec{q}, \vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Example: Formula

	Election	Lost	Obama
D1	1	1	1
D2	0	1	0
D3	0	1	1
Q	**0**	**0**	**1**

$$Sim(Q, D1) = \frac{\sum_{i=1}^{3} Q_i D1_i}{\sqrt{\sum_{i=1}^{3} Q_i^2}\sqrt{\sum_{i=1}^{3} D1_i^2}}$$

$$Sim(Q, D2) = \frac{\sum_{i=1}^{3} Q_i D2_i}{\sqrt{\sum_{i=1}^{3} Q_i^2}\sqrt{\sum_{i=1}^{3} D2_i^2}}$$

$$Sim(Q, D3) = \frac{\sum_{i=1}^{3} Q_i D3_i}{\sqrt{\sum_{i=1}^{3} Q_i^2}\sqrt{\sum_{i=1}^{3} D3_i^2}}$$

Example: Instantiation

	Election	Lost	Obama
D1	1	1	1
D2	0	1	0
D3	0	1	1
Q	**0**	**0**	**1**

$$\text{Sim}(Q,D1) = \frac{0*1 + 0*1 + 1*1}{\sqrt{1} * \sqrt{3}} = 0.577$$

$$\text{Sim}(Q,D2) = \frac{0*0 + 0*1 + 1*0}{\sqrt{1} * \sqrt{1}} = 0.000$$

$$\text{Sim}(Q,D3) = \frac{0*0 + 0*1 + 1*1}{\sqrt{1} * \sqrt{2}} = 0.707$$

Terms rarity

- Common terms are less informative than rare terms
- Consider a query like "*good refrigerators*"
  - With TF (term frequency) we consider the words frequency in each document
  - However, *good* is a very common word, that can be found in most documents, while *refrigerators* is less common and thus more informative for relevance
  - We **should weight more the rare words** than the common words
- We will use **inverse document frequency** (**idf**) to capture this, which in turn is based on document frequency (**df**)

DF: Document Frequency

- df_t is the document frequency of *t*: the number of documents that contain *t*
  - Note: despite its name is *document* frequency, it involves the **whole collection of documents!**
  - It measures **how common a term is in the whole collection of documents**
- df_t is an **inverse** measure of the informativeness of *t*
  - The **more common** is a word, the **less informative** will be for relevance (extreme case: stop words)
  - We use the **df** to formulate the **inverse** concept, in order to express the **rarity** of a word

IDF: Inverse Document Frequency

- We define the **inverse document frequency** (idf) of term *t* by

$$\text{idf}_t = \log_{10}(\text{N}/\text{df}_t)$$

  - *N* is the number of documents in the collection
- We use **log** (N/df_t) instead of N/df_t to smooth the effect of idf, for example:
  - a common word like *"home"* can have a frequency **1 million bigger** than a rare word such as *"fenestration"*
  - we apply the logarithm to flatten this **exponential** phenomenon of linguistics (Zipf's law), otherwise very common words would have an idf of zero

Effect of IDF on Ranking

- Does idf have an effect on ranking for one-term queries, like
  - 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
- IDF is never used alone, instead is used together with the TF for a better estimation of document relevance

TF-IDF 1/2

- The tf-idf weight of a term is the product of its tf weight and its idf weight

$$w_{t,d} = tf_{t,d} \times \log_{10}(N/df_t)$$

- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, **tf x idf**

TF-IDF 2/2

$$w_{t,d} = tf_{t,d} \times \log_{10}(N/df_t)$$

- The tf-idf weight of a term:
  - Increases with the number of occurrences within a document (**tf**)
  - Increases with the rarity of the term in the collection (**idf**)

Retrieval Models

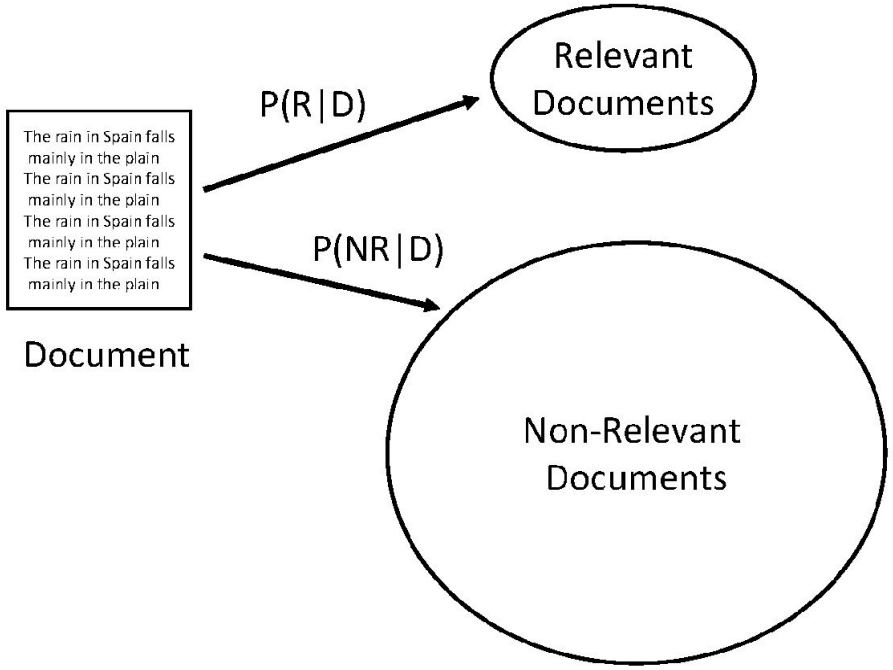
- **Classic models**
  - Boolean retrieval
  - Vector Space model
- **Probabilistic Models**
  - BM25
  - Language models

- Progress in retrieval models has corresponded with improvements in effectiveness

Probabilistic Models

- The Probability Ranking Principle in IR, Robertson (1977)
  - The ranking is given by a **probability of relevance**
  - Probability is estimated as accurately as possible using the available data → best effectiveness

"[...] If the **ranking of the documents** is **in order of decreasing probability of relevance** (where the probabilities are estimated as accurately as possible from the available data) **the overall effectiveness will be the best** that is obtainable on the basis of those data."



Okapi BM25

- The *BM25 weighting scheme*, often called *Okapi weighting*, after the system in which it was first implemented, was developed as a way of building a probabilistic model sensitive to
  - **term frequency**
  - **term rareness** in corpus (similarly to IDF)
  - **document length**
- BM25 became a popular and effective ranking algorithm while relying on probabilistic foundations
- The BM25 term weighting formulas have been used quite widely and quite successfully across a range of collections and search tasks

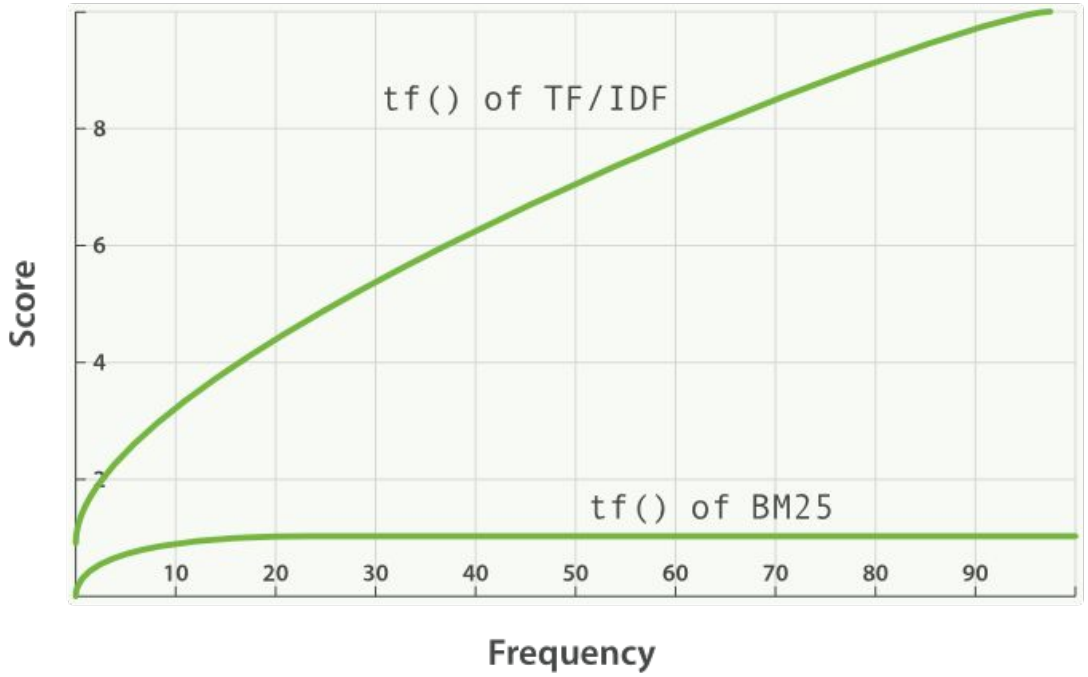
BM25 Formula

$$\mathbf{BM25}(D,Q) = \sum_{i=1}^{n} \mathbf{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\mathbf{avgdl}}\right)}$$

- q_i is a query term
- $f(q_i,D)$ is q_i's term frequency in the document D
- D is the length of the document
- avgdl is the average document length in the text collection
- k_1 and b are free parameters, usually chosen empirically (common values are k_1=2.0 and b=0.75)
- IDF(q_i) is the IDF for q_i

Okapi BM25: TF saturation

- In TF-IDF, a document similarity score could reach very high values if the term frequency of the query term is very high
- In Okapi BM25, the similarity score is always comprised between 0 and 1
  - At a certain point, highest values of term frequency do not affects the score



Okapi BM25: parameters

- Okapi BM25 has two parameters:
  - **k1**: this parameter controls **how quickly an increase in term frequency results in term-frequency saturation** (i.e., the slope of “tf() of BM25” curve)
    - The default value is 1.2
    - Lower values result in quicker saturation
    - Higher values in slower saturation
  - **b**: this parameter controls **how much effect document length normalization should have**
    - The default value is 0.75
    - A value of 0.0 disables normalization completely
    - A value of 1.0 normalizes fully

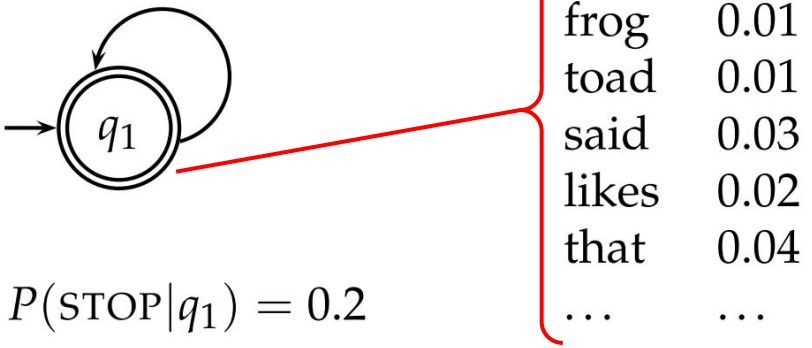
Language Model

- A statistical language model assigns a probability to a sequence of m words $P(w_1,\ldots,w_m)$ by means of a probability distribution
- A separate language model is associated with each document in a collection. Documents are ranked based on the probability of the query Q in the document's language model $P(Q \mid M_d)$.
- Unigram language model
  - probability distribution over the words in a language
  - generation of text consists of pulling words out of a “bucket” according to the probability distribution and replacing them
- N-gram language model
  - some applications use bigram and trigram language models where probabilities depend on previous words

Language Models for IR

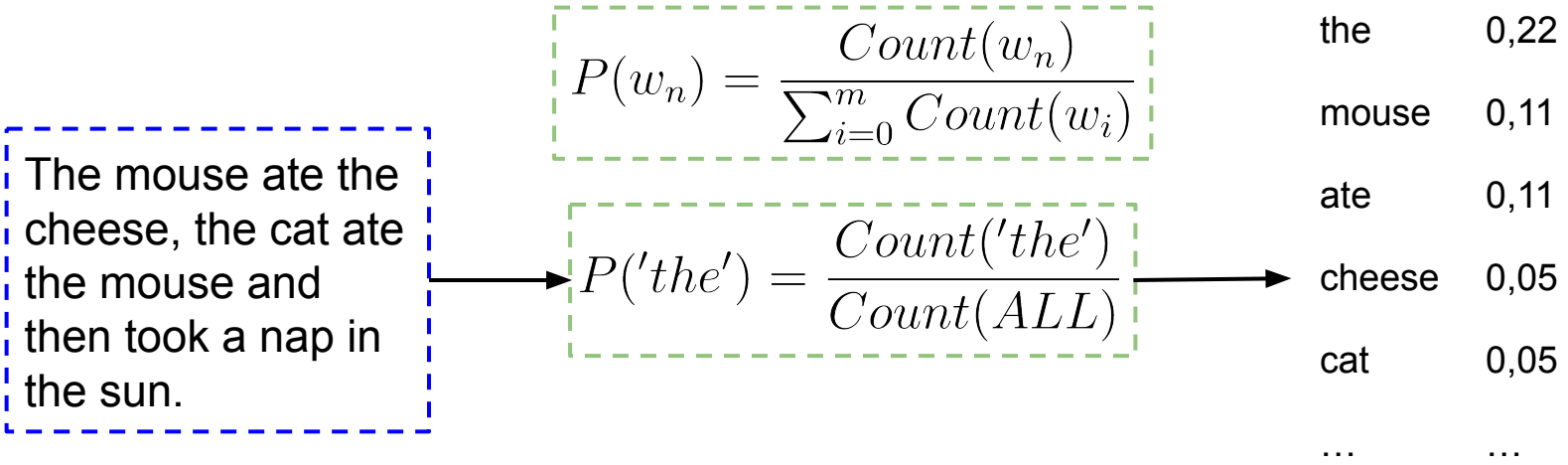
- A document is a **good match** to a query if the **document model** is likely to **generate** the **query**, which will in turn happen if the document contains the **query words often**
- A **language model** is a **finite automaton** with a **probability measure** over strings it can produce

The sum of all possible terms plus the possibility of stopping is 1



A Simple Probability Measure

- The easiest way to compute the probability of a given word in an input is to **count its occurrences** and divide that number by **the total words** in the input



- This kind of Language Model is called **Unigram** model and the probability of a word only depends on its frequency in the input text

$$P(s) = P(w_1, \ldots, w_m)$$

Unigram Language Models

- To find the probability of generating a sentence S composed of n words, we multiply the probability of each word being generated

$$P(S) = P(w_0)P(w_1)...P(w_n)$$

- One important caveat of this model is that any combination of the same words has the same probability of occurring in the model (*e.g.*, “two times” and “times two” have the same probability of occurring)
- This is the easiest model to compute and use but it’s easy to see that we can improve on this idea of “independently generated” language

Bigram LM Example 1/2

- Bigram in *“Espresso is awesome, and user-friendly”*:
  - *“espresso is”*
  - *“is awesome”*
  - *“and user”*
  - *“user friendly”*
- The documents’ sentences are split to create the **dictionary of bigrams and unigrams**
- Counting to compute the probability of each bigram that allow to find the **probability of the next word**
- **N-grams**: A large value for n makes the calculation quite intensive. In contrast, with too small value, long-term dependencies are not taken into account

Bigram LM Example 2/2

- Bigrams are the easiest n-grams to visualize because the probability of co-occurrence of two words is seen as a table with all the words in the input as both rows and columns.
- As an example, we take the Berkeley Restaurant Project Sentences Corpus with 9222 sentences and we focus on the probability of bigrams between 8 words

We calculate all probabilities using the same formula as all other n-grams

$$P(want|i) = \frac{Count('i'\ 'want')}{Count('i')}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

N-grams Language Models

- The words in a **Unigram Language Model** have **no correlation** with each other, they are treated as **separate events**. In a **real document** though, **some words** appear in succession **more often than others** (e.g. “Merry Christmas”)
- If we want to model this fact we can use **N-grams Language Models** that condition on the **previous n-1 terms** (w_i is the word at index i):

$$P(w_i...w_n) = \prod_{i=2}^{n+1} p(w_i|w_{i-n+1}...w_{i-1})$$

- **N-Grams** are **Language Models** that correlate **n words together.** The models with **n ≥ 2** are used mostly for tasks like **speech recognition**, **spelling correction**, and **machine translation**, where you need the probability of a term conditioned on surrounding context and not for IR

Ranking using Language Models

- For each document a model is created where the probability of each word in it occurring is calculated with any n-gram represented by **u** in the formula below

$$p(w|\mathbf{u}) = \frac{c(\mathbf{u}w)}{\sum_{w'} c(\mathbf{u}w')}$$

where **u** ranges over (n−1)-grams

- When a query is issued, each model computes the probability of the query appearing as a random sample from its document. The results are then ranked in descending order of probability

LM Unigram Ranking Example

Query: "The mouse ate the cheese"

Doc 1

the	0.2
mouse	0.01
ate	0.2
cheese	0.5

P(Query) = 0.2 * 0.01 * 0.2 * 0.2 * 0.5 = **$4 * 10^{-5}$**

Doc 2

the	0.1
mouse	0.33
ate	0.01
cheese	0.2

P(Query) = 0.1 * 0.33 * 0.01 * 0.1 * 0.2 = **$6.6 * 10^{-6}$**

Doc 3

the	0.4
mouse	0.06
ate	0.1
cheese	0.4

P(Query) = 0.4 * 0.06 * 0.1 * 0.4 * 0.4 = **$3.84 * 10^{-4}$**

Final Ranking

1. Doc 3
2. Doc 1
3. Doc 2

Efficiency Aspects of Ranking

- Computing the relevance score using non boolean models can be quite computationally-intensive:
  - Document is retrieved (has a relevance score) even if not all the query terms are found in it
  - Complex relevance formulas requires additional computations to compute score
- Strategies to control the efficiency:
  - The way we access documents affects computation time (one document at a time or one term at a time?)
  - Most of the time we are not interested in the least relevant documents

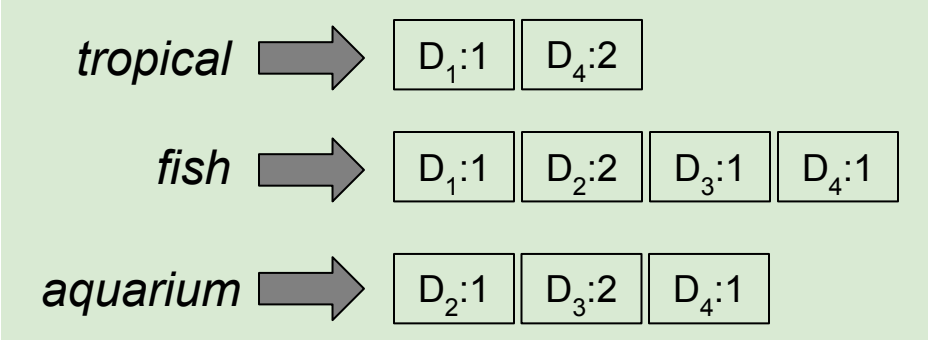
TAAT vs DAAT Techniques

- **TAAT** : “Term At A Time”
  - Scores for all docs computed concurrently, one query term at a time
- **DAAT** : “Document At A Time”
  - Total score for each doc (including all query terms) computed, before proceeding to the next
- Each has implications for how the retrieval **index** is structured and stored

TAAT: Example

Query: *tropical fish aquarium*

Inverted index:



Term frequency for each document:

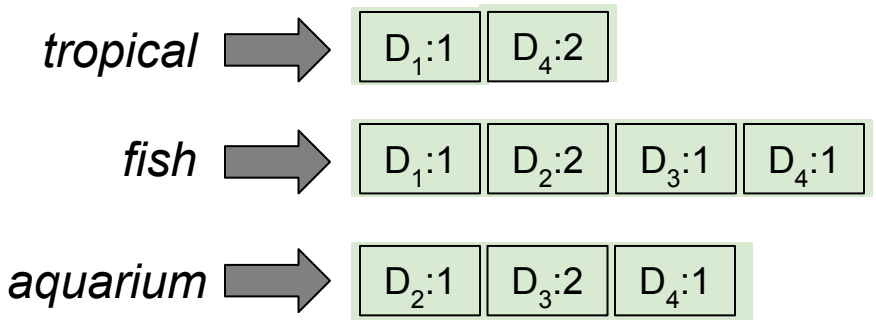
$D_1 = 1 + 1 = 2$
$D_2 = 2 + 1 = 3$
$D_3 = 1 + 2 = 3$
$D_4 = 2 + 1 + 1 = 4$

- We access one term and then move on to the next
- The complete score of a document is unknown until all query terms are processed
- We use a lot of memory to keep track of the current documents

DAAT: Example

Query: *tropical fish aquarium*

Inverted index:



Term frequency for each document:

$D_1 = 1 + 1 = 2$
$D_2 = 2 + 1 = 3$
$D_3 = 1 + 2 = 3$
$D_4 = 2 + 1 + 1 = 4$

- We fully score one document before moving to the next
- Only requires a small amount of memory (top 10)
- Easy to define if query satisfies certain conditions

Efficient Scoring

- A large fraction of the CPU work for each search query is devoted just to **compute the score** of relevance
  - Generally, we have a tight budget on latency (*e.g.,* 250ms) to answer the query, otherwise the user will not be happy
  - CPU provisioning doesn't permit exhaustively scoring every document for every query
- Basic idea: avoid scoring docs that won't make it into the top K

Safe Ranking

- The terminology “safe ranking” is used for methods that guarantee that the K docs returned are the **K absolute highest scoring** documents
  - (Not necessarily just under cosine similarity)
- Is it ok to be non-safe?
  - Yes, the results would be approximated but good enough to satisfy the user
- The goal is to run less computations while giving good results

Non-safe Ranking

- Non-safe ranking may be okay
  - Ranking function is only a proxy for user happiness
  - Documents close to top K may be just fine
- Non-safe: **Index "elimination"**
  - Only consider **high-idf (rare) query terms**, discard the common ones
  - Only consider **documents containing many query terms**, discard the ones with only few query terms
- Non-safe: **Champion lists**
  - For each term, the scores of top relevant documents are pre-computed (before any query is issued)

References

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze
Introduction to Information Retrieval
Cambridge University Press. 2008

The book is also online for free:

- HTML edition (2009.04.07)
- PDF of the book for online viewing (with nice hyperlink features, 2009.04.01)
- PDF of the book for printing (2009.04.01)