AI6122 Text Data Management & Analysis

Topic: TFIDF and Vector Space Model

Topics to be covered

- Ranked retrieval
- Scoring documents
- Weighting schemes
- Vector space scoring

Why ranked retrieval

- Boolean query: documents **either match or don't**.
  - Often result in either **too few** (=0) or **too many** (1000s) results.
    - Query 1: "*standard user dlink 650*" → 200,000 hits
    - Query 2: "*standard user dlink 650 no card found*": 0 hits
  - Good for expert users with precise understanding of their needs and the collection.
  - Good for computer programs: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users are incapable of writing Boolean queries
  - Most users don't want to wade through 1000s of results.

Query-document matching scores

- We need a way of assigning a score to a query/document pair

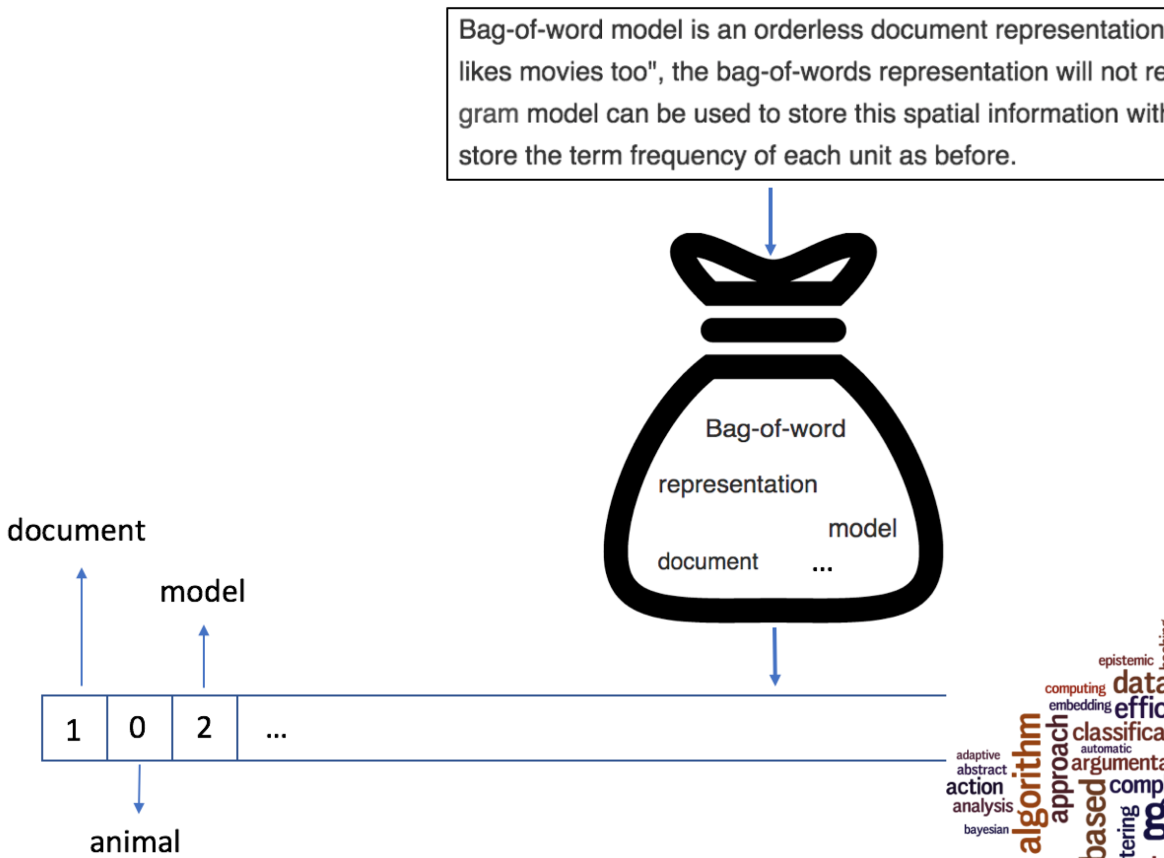
- If the query is a **one-term** query
  - If the query term does not occur in the document: score (should be) 0
    - Not considering the complicated case: e.g., laptop vs computer
  - The more frequent the query term in the document, the higher the score (should be)

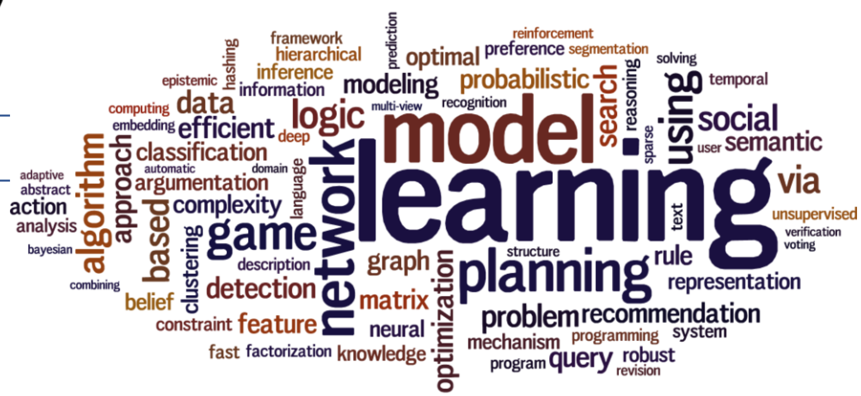
- How about multi-term queries?

“Bag of words” model

- Vector representation doesn’t consider the ordering of words in a document
  - “John is quicker than Mary”
  - “Mary is quicker than John”
  - The above two sentences have the same vectors
- This is called the **bag of words** model.
  - In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at “recovering” positional information later.
  - For now: bag of words model
  - Bag of words of a sentence/paragraph/document/collection reflects to its meaning to some extent.

Bag of words model





Source: https://www.datacamp.com/community/tutorials/lda2vec-topic-model

How to represent a document: Term frequency

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

- Term frequency in raw may not be what we want:
  – A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  – But probably not 10 times more of relevance
  – Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency **weight of term** *t* in *d* is

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

- The above is one possible definition. Example values:
  - TF: 0 → weight: 0,
  - TF: 1 → weight: 1,
  - TF: 2 → weight: 1.3,
  - TF: 10 →weight: 2,
  - TF: 1000 → weight: 4

Document frequency

- **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., arachnocentric)
  - A document containing this term is very likely to be relevant to the query arachnocentric
  - We want a high weight for rare terms like arachnocentric.

Document frequency

- **Frequent terms** are **less informative** than rare terms
- Consider a query term that is frequent in a collection
  - Example terms: *high, increase, line*
  - A document containing such a term is more likely to be relevant than a document that doesn't, but it's not a very good indicator of relevance.
- We will use document frequency (df) to measure informativeness of a term
  - Weights for words like *high, increase,* and *line* shall be higher than stop words, but lower than rare terms.

idf weight

- df_t is the document frequency of term t in a collection:
  - The number of documents that contain t in the collection
  - $\mathrm{df}_t \leq N$, the total number of documents in the collection

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

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

  - We use log (N/df_t) instead of N/df_t to smooth the effect of idf.
  - There is one *idf* value for each term t in a collection.

Effect of *idf* on ranking

- IDF has NO effect on ranking for one-term queries
  - Example query: iPhone

- IDF does affect document ranking for queries with two or more terms
  - Example query "capricious person",
  - idf weighting makes occurrences of "capricious" much more effective in document ranking than occurrences of "person".

  - How about query "Nanyang Technological University"?

Collection Frequency vs. Document frequency

- The collection frequency of *t* is the **number of occurrences** of *t* in the collection, counting multiple occurrences.

- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

  – Which word is a better search term (and should get a higher weight)?

tf-idf weighting

- The *tf-idf* weight of a term is the product of its *tf* weight and its *idf* weight.
  - Increases with the number of occurrences within a document
  - Increases with the rarity of the term in the collection

$$\mathrm{w}_{t,d} = (1+\log_{10}\mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

- 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

Score for a document given a query

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

- Simple scoring for matching documents (for now)

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

Binary → count → weight matrix

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

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Documents as vectors

- Given a vocabulary of V terms, we have a |V|-dimensional vector space
  - Terms are axes of the space
  - Documents are points or vectors in this space

- This vector space is 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.

Document ranking by their relevance to a query

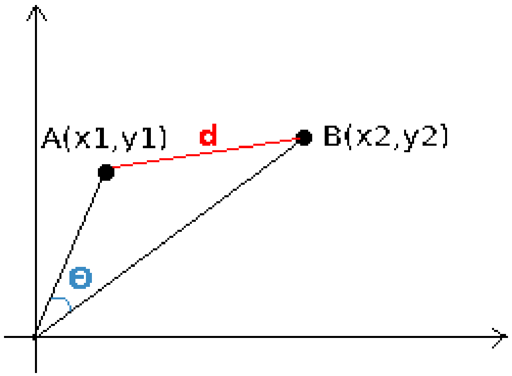
- Key idea 1: Do the same for the query:
  - Consider a query is a short document
  - represent the query as a vector in the space

- Key idea 2: Rank documents by their **proximity** to the query
  - more relevant documents are ranked higher than less relevant documents
  - proximity = similarity of vectors
  - proximity ≈ inverse of distance

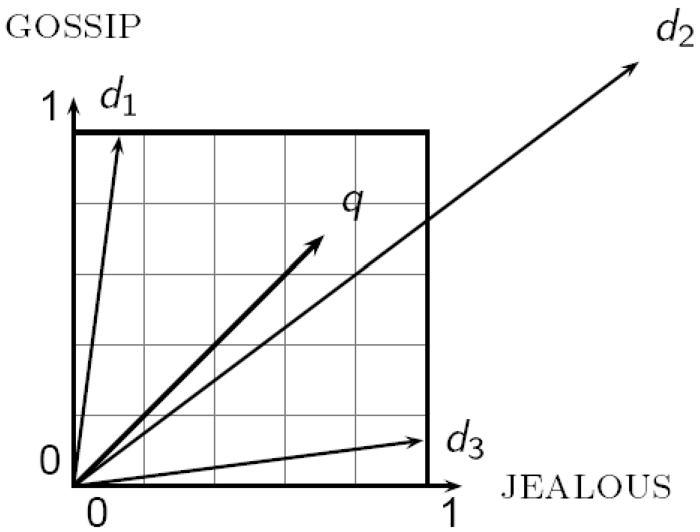
- Next question: how do we measure the similarity between two vectors

Euclidean distance is NOT a good choice

- Euclidean distance is large for vectors of different lengths.



- Example
  - Assuming the vocabulary is 2: Gossip, Jealous
  - The Euclidean distance between *q* and *d2* is large, compared with *d1* and *d3*
  - However, the distribution of terms in the query *q* and the distribution of terms in document *d2* are very similar.



- Another way of thinking:
  - take a document *d* and append it to itself. Call this document *d′*.
  - "Semantically" *d* and *d′* have the same content
  - The Euclidean distance between the two documents can be quite large

From angles to cosines

- Instead of Euclidean distance, use "angle" of vectors
  - Rank documents in <u>decreasing</u> order of the angle between query and document
  - The same as: Rank documents in <u>increasing</u> order of *cosine(query, document)*

- Cosine is a monotonically decreasing function for the interval [0º, 180º]

$$\cos(\vec{q},\vec{d}) = \frac{\overbrace{\vec{q}\bullet\vec{d}}^{\text{Dot product}}}{|\vec{q}||\vec{d}|} = \overbrace{\frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|}}^{\text{Unit vectors}} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

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

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L2 norm: $\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$

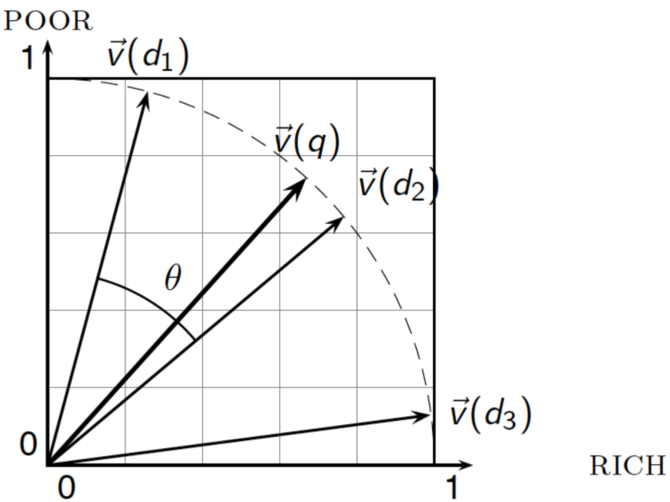
- Dividing a vector by its L2 norm makes it a unit (length) vector (on surface of unit hypersphere)

- Effect on the two documents d and d′ (d appended to itself) from earlier slide:
  - they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

Cosine for length-normalized vectors

- For **length-normalized** vectors, cosine similarity is simply the dot product (or scalar product):

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



Cosine similarity amongst 3 documents

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

- How similar are the novels
  - SaS: *Sense and Sensibility*
  - PaP: *Pride and Prejudice*, and
  - WH: *Wuthering Heights*?

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈ 0.789 × 0.832 + 0.515 × 0.555 + 0.335 × 0.0 + 0.0 × 0.0≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

Note: To simplify this example, we don’t do idf weighting.

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - \mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,*
- A very standard weighting scheme is: lnc.ltc for document.query

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
  - SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table

- A very standard weighting scheme is: lnc.ltc
  - Document: (lnc)
    - logarithmic tf (l as first character), no idf and cosine normalization
    - Leaving off idf weighting on documents is good for both efficiency and system effectiveness reasons.
  - Query: (ltc)
    - logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization …

Summary – vector space ranking

- Document ranking
  - Represent the query as a weighted tf-idf vector
  - Represent each document as a weighted tf-idf vector
  - Compute the cosine similarity score for the query vector and each document vector
  - Rank documents with respect to the query by score
  - Return the top K (e.g., K = 10) to the user

- TFIDF and Cosine similarity
  - Widely used to compute similarity between two sentences, or two documents
  - Example:
    - Construct a graph of sentences in a document
    - Find similar documents from a document collection
    - Event detection from document streams, e.g. social media