Introduction to

Informa l

Lecture 6: Scoring, Ter g and the Vector Space Model

This lecture; IIR Se .2-6.4.3

- Ranked retrieval
- Scoring documents
- Term frequency
- Collection sta
- Weighting schemes
- Vector space scoring

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs a
  - Also good for applications: A can easily consume 1000s of results.

- 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.

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: "*standard user dlink 650*" → 200,000 hits
- Query 2: "*sta* *card found*": 0 hits
- It takes a lot of skill to com query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

Ranked retrieval m

- Rather than a set of documents satisfying a query expression, in ranked retrieval models, the system returns an ordering over the (top) documents in the collection wit

- Free text queries: Rather th language of operators and expressions, query is just one or more words in a human language

- In principle, there are two separate choices here, but in practice, ranked retrieval models have normally been associated with free text queries and vice versa

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show t
  - We don't ov
  - Premise: the ranking algorithm works

Scoring as the basi ked retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection wit
- Assign a score – say in [0, 1 document
- This score measures how w ent and query "match".

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Take 1: Jaccard co t

- Recall from Lecture 3: A commonly used measure of overlap of two sets *A* and *B*
- jaccard$(A,B) = |A \cap B| \,/\, |A \cup B|$
- jaccard$(A,A) =$
- jaccard$(A,B) = 0$ if $A \cap B = 0$
- *A* and *B* don't have to be th e.
- Always assigns a number between 0 and 1.

Jaccard coefficient g example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?

- Query: *ides of*

- Document 1: *caesar died in*

- Document 2: *the long marc*

> the term *Ides of March* is best known as the date that Julius Caesar was killed in 709 AUC or 44 B.C

Issues with Jaccard ring

1 It doesn’t consider *term frequency* (how many times a term occurs in a document)

2 Rare terms in a collection are more informative than freque ’t consider this information

- We need a more sophisticated w lizing for length

Recall (Lecture 1): Binary term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	[illegible]	[illegible]	1	1	1
Calpurnia	0	[illegible]	[illegible]	0	0	0
Cleopatra	1	0	[illegible]	0	0	0
mercy	1	0	[illegible]	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^{|V|}$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	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

Bag of words mod

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the
- This is called the bag of wo
- In a sense, this is a step bac itional index was able to distinguish these two documents.
- We will look at "recovering" positional information later in this course.
- For now: bag of words model

Term frequency tf

- The term frequency $\text{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores.

- Raw term frequency is not ant:

  - A document with 10 occurre erm is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR

Log-frequency wei

- The log frequency weight of term t in d is

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

- 0 → 0, 1 → 1, … → 4, etc.

- Score for a document-quer… over terms *t* in both *q* and *d*:

- score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$

- The score is 0 if none of the query terms is present in the document.

Document frequen

- Rare terms are more informative than frequent terms
  - Recall stop words

- Consider a ter are in the collection (e.g.

- A document containing this ry likely to be relevant to the query *arachn*

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

Document frequen tinued

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document c is more likely to be relevant than a docume sn't
- But it's not a sure indicator ce.
- → For frequent terms, we want high positive weights for words like *high, increase, and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $\text{df}_t \leq N$

- We define the t frequency) of t by

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

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

Will turn out the base of the log is immaterial.

idf example, suppo million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday		
fly		
under		
the	1,000,000	

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

There is one idf value for each term *t* in a collection.

Effect of idf on ran

- Does idf have an effect on ranking for one-term queries, like
  - iPhone

- idf has no eff queries

  - idf affects the ranking of doc 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.

Collection 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.

$$w_{t,d} = (1 \quad \ldots \quad N/\mathrm{df}_t)$$

- Best known weighting sche … ation retrieval
  - Note: the “-” in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Final ranking of docs for a query

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

Binary → count → weight matrix

	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			1.51	0.25	0
Calpurnia	0			0	0	0
Cleopatra	2.85	0		0	0	0
mercy	1.51	0		0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

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

Documents as vect

- So we have a |V|-dimensional vector space
- Terms are axes of the space
- Documents ar this space
- Very high-dim ons of dimensions when you appl web search engine
- These are very sparse vectors - most entries are zero.

Queries as vectors

- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to t

- proximity = similarity of ve
- proximity ≈ inverse of dista

- Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

Formalizing vector roximity

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?

- Euclidean dist

- . . . because Euclidean dist e for vectors of different lengths.

Why distance is a a

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of ter

in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.

Use angle instead ce

- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
- “Semantically” d and d′ have the same content
- The Euclidean two documents can be quite large
- The angle between the two ts is 0, corresponding to maximal similarity.

- Key idea: Rank documents according to angle with query.

From angles to cos

- The following two notions are equivalent.
  - Rank documents in <u>decreasing</u> order of the angle between query and document
  - Rank docum cosine(query
- Cosine is a monotonically d unction for the interval [0°, 180°]

From angles to cos

- But how – *and why* – should we be computing cosines?

Length normalizati

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\left\| \rightarrow \right\| \quad \sqrt{\quad {}^{2}}$$

- Dividing a vector by its L_2 n it a unit (length) vector (on surface ersphere)

- 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(query,document)

Dot product | Unit vectors

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

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

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

Cosine for length-normalized vectors

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

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

for q, d length-normalized.

Cosine similarity ill d

Cosine similarity am [illegible] ocuments

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
[illegible]	[illegible]	7	11
gossip	[illegible]	0	6
wuthering	0	0	38

Term frequencies (counts)

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

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06		
jealous	2.00	[illegible]	[illegible]
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
	0.789	0.832	0.524
[illegible]	0.515	0.555	0.465
g[illegible]	0.335	0	0.405
[illegible]	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

Computing cosine s

tf-idf weighting ha variants

Columns headed 'n' are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

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, wit ... q, using the acronyms fro ...
- A very standard weighting ... **lnc.ltc**
  - Document: logarithmic tf (l as first character), no idf and cosine normalization

A bad idea?

  - Query: logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization …

tf-idf example: lnc.ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt						tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0			1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Exercise: what is *N*, the number of docs?

Doc length $=\sqrt{1^2+0^2+1^2+1.3^2} \approx 1.92$

Score = 0+0+0.27+0.53 = 0.8

Representation/fe erspective

- lnc.ltc
  - doc vector:
    - tf-vector
    - normalized
  - query vector
    - tf-idf-vector
    - normalized to unit length
  - score = similarity = inner product between the two vectors

Summary – vector anking

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the for the query vector and eac
- Rank documents with resp uery by score
- Return the top *K* (e.g., *K* = 10) to the user

Resources for toda ure

- IIR 6.2 – 6.4.3

- http://www. n-retrieval-tutorial/cosin ml
  - Term weighting and cosine s orial for SEO folk!