Introduction to

Information Retrieval

Lecture 6: Scoring, Term Weighting and the Vector Space Model

Contents

- Parametric and zone indexes
- Ranked retrieval
- Term Weighting schemes
- Vector space scoring

Parametric and zone indexes

- Each document has, in addition to text, some "meta-data" in fields e.g.,
  - Language = French
  - Format = pdf
  - Subject = Physics etc.
  - Date = Feb 2000
- A parametric search interface allows the user to combine a full-text query with selections on these field values e.g.,
  - language, date range, etc.

Fields → Values

User view

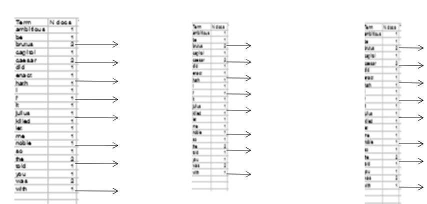
► Figure 6.1 Parametric search. In this example we have a collection with fields allowing us to select publications by zones such as Author and fields such as Language.

4

Zones

- A zone is an identified region within a doc
  - E.g., Title, Abstract, Bibliography
- Contents of a zone are free text
  - Not a "finite" vocabulary
- Indexes for each zone - allow queries like
  - find documents with merchant in the title and william in the author list and the phrase gentle rain in the body

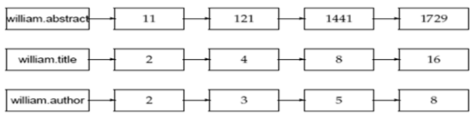
Zone indexes – simple view



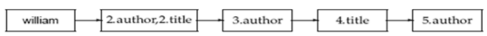
etc.

Title Author Body

Zone indexes – common view



► **Figure 6.2** Basic zone index ; zones are encoded as extensions of dictionary entries.



► **Figure 6.3** Zone index in which the zone is encoded in the postings rather than the dictionary.

Weighted zone scoring

- Given a Boolean query *q and a document d, weighted zone scoring assigns* to the pair (*q, d) a score in the interval [0, 1],*
  - *by computing a linear combination* of *zone scores*
  - *where each zone of the document contributes a Boolean* value.
- Specifically,
  - let there are L zones. Let $g_1, \ldots, g_L \in [0, 1]$ *such that* $\sum_{i=1}^{L} g_i = 1$
  - let s_i *be the* Boolean score denoting a match (or absence thereof) between *q and the ith* zone.
  - *Then,* the weighted zone score is defined to be $\sum_{i=1}^{L} g_i * s_i$

Example: Weighted zone scoring

- Query Q= shakespeare
- consider a collection in which each document has three zones: ***author, title and body***
- Suppose we set *$g_1 = 0.2$, $g_2 = 0.3$ and $g_3 = 0.5$ where g_1, g_2 and g_3 represents the **author, title and body** zone weights.*
- If the term shakespeare appear in the **title** and **body** zones but not the *author* zone of a document, the score of this document would be
- 0.8.

Ranked retrieval models

- 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
- Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language

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	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	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: 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 model

- 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 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 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. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term 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 weighting

- The log frequency weight of term t in d is

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

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- score $= \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- → We want a high weight for rare terms.

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*
  - $\text{df}_t \le N$
- We define the idf (inverse document frequency) of *t* by

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

  - 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, suppose *N* = 806791

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

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

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

tf-idf weighting

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

$$\text{tf-idf}_{t,d} = \log(1+\text{tf}_{t,d}) \times \log_{10}(N/\text{df}_t)$$

- 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
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Binary → count → tf-idf 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	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

Each document is now represented by a real-valued vector of tf-idf

Score for a document given a query

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

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

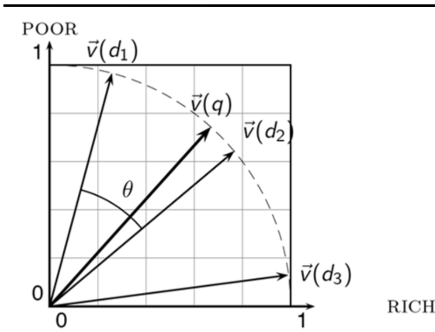
The Vector Space Model for Scoring

- The set of documents in a collection may be viewed as a set of vectors in a 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.

Vector Similarity

- How do we quantify the similarity between two documents in this vector space?
- A first attempt : the magnitude of the vector difference between two document vectors.
  - Drawback: two documents with very similar content can have a significant vector difference simply because one is much longer than the other.
- **Solution** to compensate for the effect of document length is to compute the *cosine similarity*.

Cosine similarity illustrated



From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0°, 180°]

cosine(document,document)

DOT PRODUCT

$$\mathrm{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \quad (2)$$

EUCLIDEAN LENGTH

The dot product $\vec{x} \cdot \vec{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$

The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$.

Cosine for length-normalized vectors

The effect of the denominator of Equation (2) is thus to *length-normalize* the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ to unit vectors $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and $\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$.

Then we can rewrite the previous equation as:

$$\mathrm{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \quad (3)$$

Example: N= 1,000,000

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

Term	DF
auto	5000
best	50000
car	10000
insurance	1000

Example: N= 1,000,000

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

Term	Query	Document	Prod
auto			
best			
car			
insurance			

Example: N= 1,000,000

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

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	2.3	0.47	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.53	1	1	2.0	0.41	0.22
insurance	1	1	1000	3.0	3.0	0.79	2	1.3	3.9	0.80	0.63

Query length = $\sqrt{0^2 + 1.3^2 + 2^2 + 3^2} \approx 3.8$

Doc length = $\sqrt{2.3^2 + 0^2 + 2^2 + 3.9^2} \approx 4.9$

Score = 0+0+0.22+0.63 = 0.85

Example: N= 1,000,000

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

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	2.3	0.47	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.53	1	1	2.0	0.41	0.22
insurance	1	1	1000	3.0	3.0	0.79	2	1.3	3.9	0.80	0.63

Query length = $\sqrt{0^2 + 1.3^2 + 2^2 + 3^2} \approx 3.8$

Doc length = $\sqrt{2.3^2 + 0^2 + 2^2 + 3.9^2} \approx 4.9$

Score = 0+0+0.22+0.63 = 0.85

Introduction to

Information Retrieval

Lecture 8: Evaluation IR systems

Measures for a search engine

- How fast does it index
  - Number of documents/hour
  - (Average document size)
- How fast does it search
  - Latency as a function of index size

  } Efficiency
- Quality of results
  - Precision
  - Recall
  - F-measure

  } Effectiveness
- Expressiveness of query language
  - Ability to express complex information needs

  } Usability

Evaluating an IR system

- Note: the **information need** is translated into a **query**
- Relevance is assessed relative to the **information need** *not* the **query**
- E.g., Information need: *I'm looking for information on whether travelling by train from Cairo to Assuit is more effective than flying.*
- Query: ***travelling by train from Cairo to Assuit effective***
- Evaluate whether the doc addresses the information need, not whether it has these words

Standard relevance benchmarks

- TREC - National Institute of Standards and Technology (NIST) has run a large IR test bed for many years
- Reuters and other benchmark doc collections used
- "Retrieval tasks" specified
  - sometimes as queries
- Human experts mark, for each query and for each doc, Relevant or Nonrelevant
  - or at least for subset of docs that some system returned for that query

Unranked retrieval evaluation: Precision and Recall

- **Precision**: fraction of retrieved docs that are relevant = (relevant retrieved / retrieved)
- **Recall**: fraction of relevant docs that are retrieved

  = (relevant retrieved/relevant)

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp/(tp + fp)
- Recall R = tp/(tp + fn)

Should we instead use the accuracy measure for evaluation?

- Given a query, an engine classifies each doc as "Relevant" or "Nonrelevant"
- The **accuracy** of an engine: the fraction of these classifications that are correct
  - (tp + tn) / (tp + fp + fn + tn)
- **Accuracy** is a commonly used evaluation measure in machine learning classification work

Precision/Recall

- You can get **high recall** (but low precision) by **retrieving all docs for all queries**!
- Recall is a non-decreasing function of the number of docs retrieved
- In a good system, precision decreases as either the number of docs retrieved or recall increases
  - This is not a theorem, but a result with strong empirical confirmation

A combined measure: *F*

- Combined measure that assesses precision/recall tradeoff is **F measure** (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

- People usually use balanced F_1 measure
  - i.e., with $\alpha = ½$

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

Evaluating ranked results

- Evaluation of ranked results:
  - The system can return any number of results
  - By taking various numbers of the top returned documents (levels of recall), the evaluator can produce a *precision-recall curve*

Averaging over queries

- A precision-recall graph for one query isn't a very sensible thing to look at
- You need to average performance over a whole bunch of queries.
- But there's a technical issue:
  - Precision-recall calculations place some points on the graph

Typical (good) 11 point precisions