




Inspire…Educate…Transform.

3. Relevance Ranking

Dr. Manish Gupta

Sr. Mentor – Academics, INSOFE

Adapted from http://web.stanford.edu/class/cs276/handouts/lecture6-tfidf.ppt

Course Content

- Collection of three main topics of high recent interest.
  - Search engines (Crawling, Indexing, Ranking)
    - Language Modeling
    - Text Indexing and Crawling
    - **Relevance Ranking**
    - Link Analysis Algorithms
  - Text Processing (NLP, NER, Sentiments)
    - Natural Language Processing
    - Named Entity Recognition
    - Sentiment Analysis
    - Summarization
  - Social networks (Properties, Influence Propagation)
    - Social Network Analysis
    - Influence Propagation in Social Networks

Today's Agenda

- **Need for Relevance Ranking**

CSE 7306c

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 and the collection.
  - Also good for applications: 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.
    - This is particularly true of web search.

Problem with Boolean search: Feast or Famine

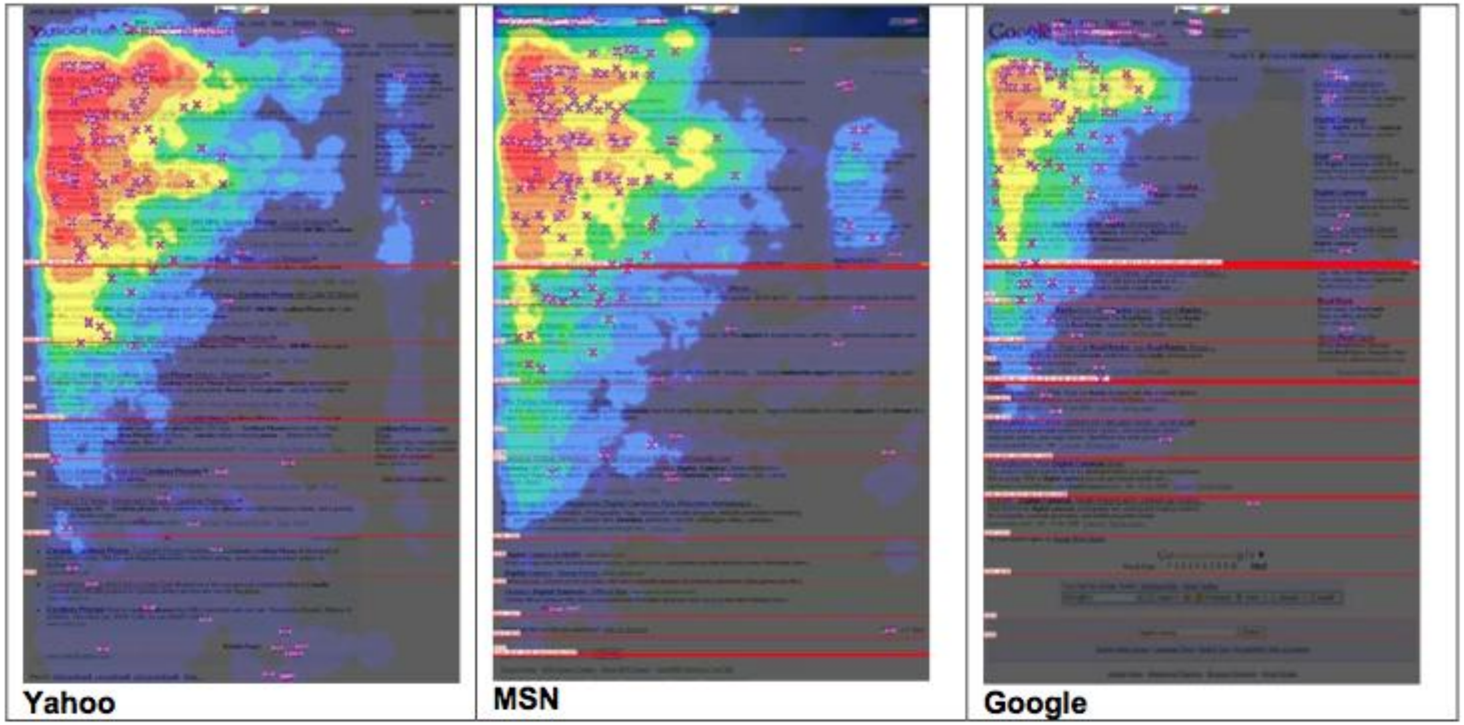
- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: "*standard user dlink 650*" → 200,000 hits
- Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

CSE 7306c

Feast or Famine: OK for 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
  - Indeed, the size of the result set is not an issue
  - We just show the top k ($\approx$ 10) results
  - We don't overwhelm the user
  - Premise: the ranking algorithm works. Is it true?

Eye Tracking Study on Search Results



http://blog.mediative.com/en/2011/08/31/eye-tracking-google-through-the-years/

CSE 7306c

Scoring as the Basis of Ranked 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 with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match."

Query-Document Matching Scores

- 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.
- First take: Jaccard coefficient?

Issues with Jaccard for Scoring

- It doesn't consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length

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 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

Today's Agenda

- Need for Relevance Ranking
- **TF and IDF**

CSE 7306c

Term Frequency TF

- The term frequency $\mathrm{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- Relevance does not increase proportionally with term frequency.
- So use log frequency weighting
- The log frequency weight of term t in d is $w_{td} = \log_{10} tf_{td}$ if $tf_{td} > 0$; else it is 0.
- Score for a document-query pair: sum over terms t in both q and d
  - $score = \sum_{t \in q \cap d} (\log_{10} tf_{td})$

CSE 7306c

Inverse Document Frequency IDF

- Frequent terms are less informative than rare terms
- df_t is the document frequency of t: the number of documents that contain t
  - $df_t \leq N$
- $idf_t = \log_{10}\left(\frac{N}{df_t}\right)$
- 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 Weighting

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

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

- 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
  - ...

CSE 7306c

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	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 weights $\in \mathbb{R}^{|V|}$

Ranking in Vector Space Model

- Represent both query and document as vectors in the |V|-dimensional space
- Use cosine similarity as the similarity measure
  - Incorporates length normalization automatically (longer vs shorter documents)

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

- 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

CSE 7306c

Summary – Vector Space 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

CSE 7306c

Today's Agenda

- Need for Relevance Ranking
- TF and IDF
- **Vector Space Model**

Efficient Cosine Ranking

- Find the K docs in the collection "nearest" to the query $\Rightarrow$ K largest query-doc cosines.
- Efficient ranking
  - Computing a single cosine efficiently.
  - Choosing the K largest cosine values efficiently.
    - Can we do this without computing all N cosines?
    - We don't need to totally order all docs in the collection
    - Let J = number of docs with nonzero cosines
      - We seek the K best of these J
    - Use heap for selecting top K

CSE 7306c

Index Elimination

- Basic algorithm only considers docs containing at least one query term
- Take this further
  - Only consider high-idf query terms
  - Only consider docs containing many query terms

Champion Lists

- Precompute for each dictionary term *t*, the *r* docs of highest weight in *t*'s postings
  - Call this the <u>champion list</u> for *t*
  - (aka <u>fancy list</u> or <u>top docs</u> for *t*)
- Note that *r* has to be chosen at index time
- At query time, only compute scores for docs in the champion list of some query term
  - Pick the *K* top-scoring docs from amongst these

CSE 7306c

Static Quality Scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document
- Examples of authority signals
  - Wikipedia among websites
  - Articles in certain newspapers
  - A paper with many citations
  - (Pagerank)
- Assign to each document a *query-independent* quality score in [0,1] to each document *d*
  - Denote this by *staticQuality(d)*

CSE 7306c

Net Score

- Consider a simple total score combining cosine relevance and authority
- net-score(q,d) = *staticQuality(d)* + cosine(q,d)
  - Can use some other linear combination than an equal weighting
  - Indeed, any function of the two "signals" of user happiness
  - Now we seek the top K docs by net score

CSE 7306c

Top *K* by Net Score – Fast Method

- First idea: Order all postings by *staticQuality(d)*
- Why? Under *staticQuality*(d)-ordering, top-scoring docs likely to appear early in postings traversal
- Key: this is a common ordering for all postings
- Thus, can concurrently traverse query terms' postings for
  - Postings intersection
  - Cosine score computation

Today's Agenda

- Need for Relevance Ranking
- TF and IDF
- Vector Space Model
- **Evaluation Metrics for Ranking**

Measures for a Search Engine

- How fast does it index
  - Number of documents/hour
- How fast does it search
  - Latency as a function of index size
- Expressiveness of query language
  - Ability to express complex information needs
  - Speed on complex queries
- Uncluttered UI

Unranked Retrieval Evaluation: Prec and Recall

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

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision: $P = tp/(tp + fp)$
- Recall: $R = tp/(tp + fn)$
- F measure: $F = \frac{2PR}{P+R}$

CSE 7306c

Normalized Discounted Cumulative Gain

- Fix query q
- Relevance level of document ranked j wrt q be $r_q(j)$
- $r_q(j)$=0 means totally irrelevant
- Response list is inspected up to rank L
- Discounted cumulative gain for query q is

cumulative

gain

$$\mathrm{NDCG}_q = Z_q \sum_{j=1}^{L} \frac{2^{r_q(j)} - 1}{\log(1 + j)}$$

normalized

rank discount

- Z_q is a normalization factor that ensures the perfect ordering has $\mathrm{NDCG}_q = 1$
- Overall NDCG is average of NDCG_q over all q
- No notion of recall, only precision at low ranks

CSE 7306c

Take-away Messages

- Binary retrieval is not enough. We need relevance ranking.
- TF and IDF are the basis of all ranking schemes.
- Vector Space Model and how to make ranking efficient.
- Evaluation metrics for ranking.

Further Reading

- Chapters 6,7,8 of [Manning-Raghavan-Schuetze book](http://nlp.stanford.edu/IR-book/)
  - http://nlp.stanford.edu/IR-book/

CSE 7306c

International School of Engineering

Plot 63/A, 1st Floor, Road # 13, Film Nagar, Jubilee Hills, Hyderabad - 500 033

For Individuals: +91-9502334561/63 or 040-65743991

For Corporates: +91-9618483483

Web: http://www.insofe.edu.in

Facebook: https://www.facebook.com/insofe

Twitter: https://twitter.com/Insofeedu

YouTube: http://www.youtube.com/InsofeVideos

SlideShare: http://www.slideshare.net/INSOFE

LinkedIn: http://www.linkedin.com/company/international-school-of-engineering