Text Processing on the Web

Week 2

Introduction to Information Retrieval and the Vector Space Model

The material for these slides are borrowed heavily from the precursor of this course by Tat-Seng Chua as well as slides from the accompanying recommended texts from Baldi et al., Larson and Hearst and Manning et al.

- Last week: HTTP / Web nuances
- Unfinished: The web as a graph: size and evolution models (save for Session w/ Tutorial 0)

Outline

- What is IR?
- TF.IDF
- Relevance Feedback
- IR Evaluation

Text Database

Different kinds of text in “Text Processing”

- Free Text - unstructured text, unlimited vocabulary. E.g., natural language text
- Structured Text - Delimited text into fields, constituting attribute value pairs. E.g, database of strings
- Semi Structured Text - Latent structure in text, but not necessarily coded in a regular style. E.g., product web pages

What is the appropriate treatment for each type of text?

Levels of Text Processing Systems

Information Retrieval

Dialog Systems

Question Answering

Information Extraction

Natural Language Processing

String Matching

More Understanding

Less Understanding

Exercise: Map these processing systems to the line below and justify

Text Analysis Example

Photo credit: markehr

Singapore Flyer

Singapore Flyer Pte Ltd 30 Raffles Avenue, #01-07 Singapore 039803
Telephone: (65) 6854 5200 Fax: (65) 6339 9167

Singapore Flyer is the world's largest observation wheel. Standing at a stunning 165m from the ground, the Flyer offers you breathtaking, panoramic views of the Marina Bay, our island city and beyond. There's also a wide range of shops, restaurants, activities and facilities. READ MORE >>

- Information Units
  - IR: terms: raffles x 1; Singapore x 3; pte x 1 …
  - IE: info units: Singapore Flyer, Raffles Avenue, Marina Bay, (65) 6854-5200 …
    and their relations
  - QA: Which is the nearest MRT to Singapore Flyer?
    Answer: City Hall MRT
  - NLP: *understanding the contents*

Information Retrieval in a nutshell

Information Needs

Text Database

Query

Matching

Indexed Doc. Representation

Ranked List

Exercise: Where's the arrows?

Doc Representation

Sad but true

Query and documents seen as a <u>bag of words</u>

Matching is done by comparing these BoWs

How do we get to a BoW given a text?

Let's look at unstructured text first:

- Tokenization - not all languages have spaces to delimit
  - what about phrases like GermanNounCompounds
  - HTML structure can help to recover latent semi structure but is not guaranteed to be well formed

Doc Representation

- Stemming - recover stem for agglutinative languages
  - For English: Porter and Lovins stemmer: uses 5 iterations to strip suffixes. Does not necessarily result in a word
  - What's a "stem" in CJK?
- Case Folding - combine the same word in different cases: next NEXT Next NeXT
- Stop Words - remove frequent words that are not used in queries.

Which of 2 of these three attack the same problem?
What is this problem?

Term Specific Weighting

Xxxxxxxxxxxxxxx IBM xxxxxxxxxxxx xxxxxxxxx xxxxxxxxxxxx IBM xxxxxxx xxxxxxxxxxx xxxxxxxxx Apple. Xxxxxxxxxxx xxxxxxxxxxx IBM xxxxxxxxx. Xxxxxxxxxxx xxxxxxxxx Compaq. Xxxxxxxxx xxxxxxx IBM.

- We call this Term Frequency

  although this is really just a count

- Forms of TF_{ij} = N_{ij}

  $1+\ln(N_{ij})$

  $N_{ij}/\max(N_i)$

Document Specific Weighting

- Which of these tells you more about a doc?
  - 10 occurrences of *hernia*?
  - 10 occurrences of *the*?
- Would like to attenuate the weight of a common term
  - But what is “common”?
- Suggest looking at collection frequency (*cf*)
  - The total number of occurrences of the term in the entire collection of documents

Document frequency

- But document frequency (*df*) may be better:
- *df* = number of docs in the corpus containing the term

Word	*cf*	*df*
ferrari	10422	17
insurance	10440	3997

- Document/collection frequency weighting is only possible in known (static) collection.
- So how do we make use of *df* ?

This is tf.idf

- tf.idf measure combines:
  - term frequency (*tf*)
    - or *wf*, some measure of term density in a doc
  - inverse document frequency (*idf*)
    - measure of informativeness of a term: its rarity across the whole corpus
    - could just be raw count of number of documents the term occurs in ($idf_i = 1/df_i$)
    - but by far the most commonly used version is:

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

- Justified as optimal weight w.r.t relative entropy

Documents as vectors

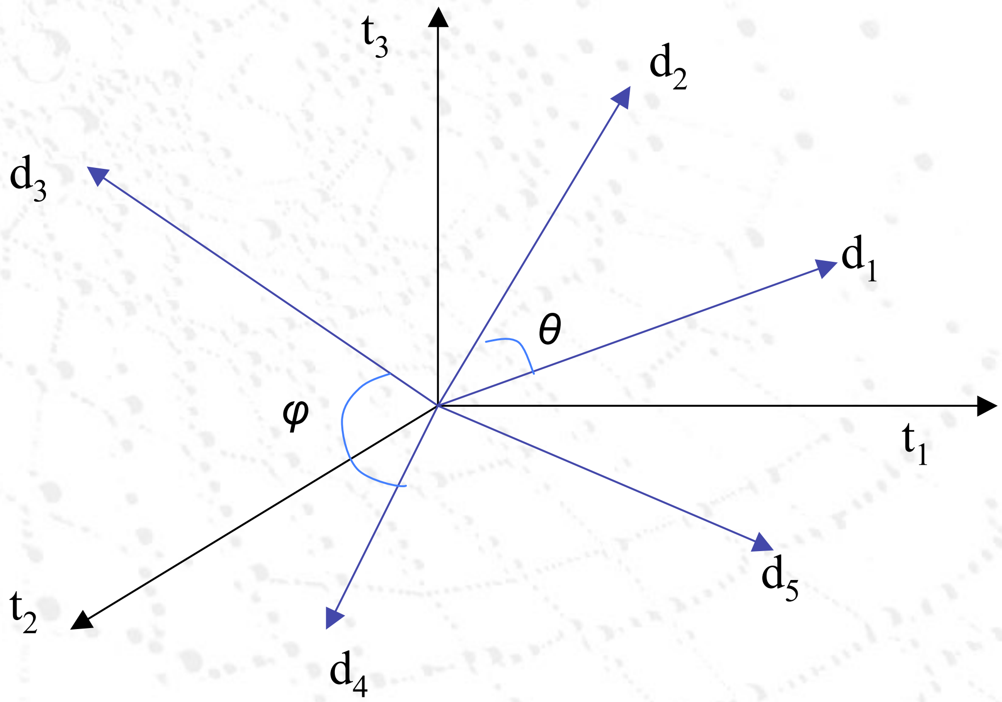
- Each doc *j* can now be viewed as a vector of *tf x idf* values, one component for each term

- So we have a vector space
  - terms are axes
  - docs live in this space
  - even with stemming, may have 20,000+ dimensions

Why turn docs into vectors?

- First application: Query-by-example
  - Given a doc *d*, find others “like” it.
- Now that *d* is a vector, find vectors (docs) “near” it.

Intuition



Postulate: Documents that are “close together” in the vector space talk about the same things.

Desiderata for proximity

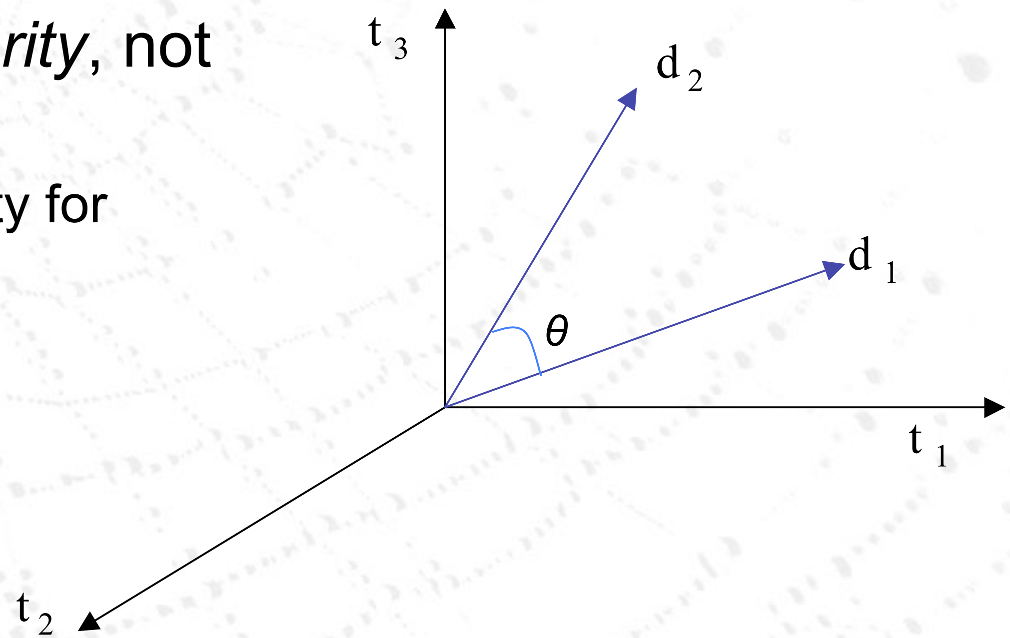
- If d_1 is near d_2, then d_2 is near d_1.
- If d_1 near d_2, and d_2 near d_3, then d_1 is not far from d_3.
- No doc is closer to d than d itself.

First cut

- Idea: Distance between d_1 and d_2 is the length of the vector $|d_1 - d_2|$.
  - Euclidean distance
- Why is this not a great idea?
- We still haven't dealt with the issue of length normalization
  - Short documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at *angles* instead

Cosine similarity

- Distance between vectors d_1 and d_2 *captured* by the cosine of the angle *x* between them.
- Note – this is *similarity*, not distance
  - No triangle inequality for similarity.



Cosine similarity

- A vector can be *normalized* (given a length of 1) by dividing each of its components by its length – here we use the L_2 norm

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- This maps vectors onto the unit sphere:
- Then, $\left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}} = 1$
- Longer documents don't get more weight

Cosine similarity

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right| \left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors.

Normalization

Normalized vectors

- For normalized vectors, the cosine is simply the dot product:

$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$

Example

- Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights. tf* weights

	SaS	**PaP**	**WH**
affection	**115**	**58**	**20**
jealous	**10**	**7**	**11**
gossip	**2**	**0**	**6**

	SaS	**PaP**	**WH**
affection	**0.996**	**0.993**	**0.847**
jealous	**0.087**	**0.120**	**0.466**
gossip	**0.017**	**0.000**	**0.254**

- cos(SAS, PAP) = .996 x .993 + .087 x .120 + .017 x 0.0 = 0.999
- cos(SAS, WH) = .996 x .847 + .087 x .466 + .017 x .254 = 0.889

Cosine similarity exercise

- *Exercise: Rank the following by decreasing cosine similarity. Assume tf.idf weighting:*
  - Two docs that have only frequent words ***(the, a, an, of)*** in common.
  - Two docs that have no words in common.
  - Two docs that have many rare words in common ***(wingspan, tailfin).***

Phrase queries

- Running multiple queries
  - Backoff to n-1 gram in case of too few results
    1. “A B C”
    2. “A B”, “B C”
    3. A, B, C
- Proximity as window *w* between term occurrences
  - Prefer the window to be smaller

Break time

- Watch the Corp Comm video

NUS School of Computing Public Symposium

(comprising two talks)

20 August 2008, 4pm to 5.30pm
SR1, COM1 Level 2
Register at: https://register.comp.nus.edu.sg/corpcomm4

NUS School of Computing
10 Years as a Faculty & 33 Years of Computing Excellence

Google: A Computer-Science Success Story
Considering Mathematical Groundwork, Pragmatics
Remaining Challenges
by Jeffrey Ullman
Stanford W Ascherman Professor of Computer Science (Emeritus)

Why Many High-paying Jobs of the Future Can Benefit from a Good University Education in Computing
by H T Kung
William H Gates Professor of Computer Science & Electrical Engineering
Harvard School of Engineering and Applied Sciences

Relevance Feedback and IR Evaluation

Relevance Feedback

- Main Idea:
  - Modify existing query based on relevance judgements
    - Extract terms from relevant documents and add them to the query
    - and/or re-weight the terms already in the query
  - Two main approaches:
    - Automatic (pseudo-relevance feedback)
    - Users select relevant documents
  - Users/system select terms from an automatically-generated list
- Will return to this later: clickstreams in web search engines

We focus on this case

Relevance Feedback

- Usually do both:
  - expand query with new terms
  - re-weight terms in query

- There are many variations
  - Usually positive weights for terms from relevant docs
  - Sometimes negative weights for terms from non-relevant docs
  - Select terms sometimes by requiring them to match query in addition to document

Rocchio Method

$$Q_1 = Q_0 + \beta \sum_{i=1}^{n_1} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{n_2} \frac{S_i}{n_2}$$

where

$Q_0 =$ the vector for the initial query

$R_i =$ the vector for the relevant document i

$S_i =$ the vector for the non-relevant document i

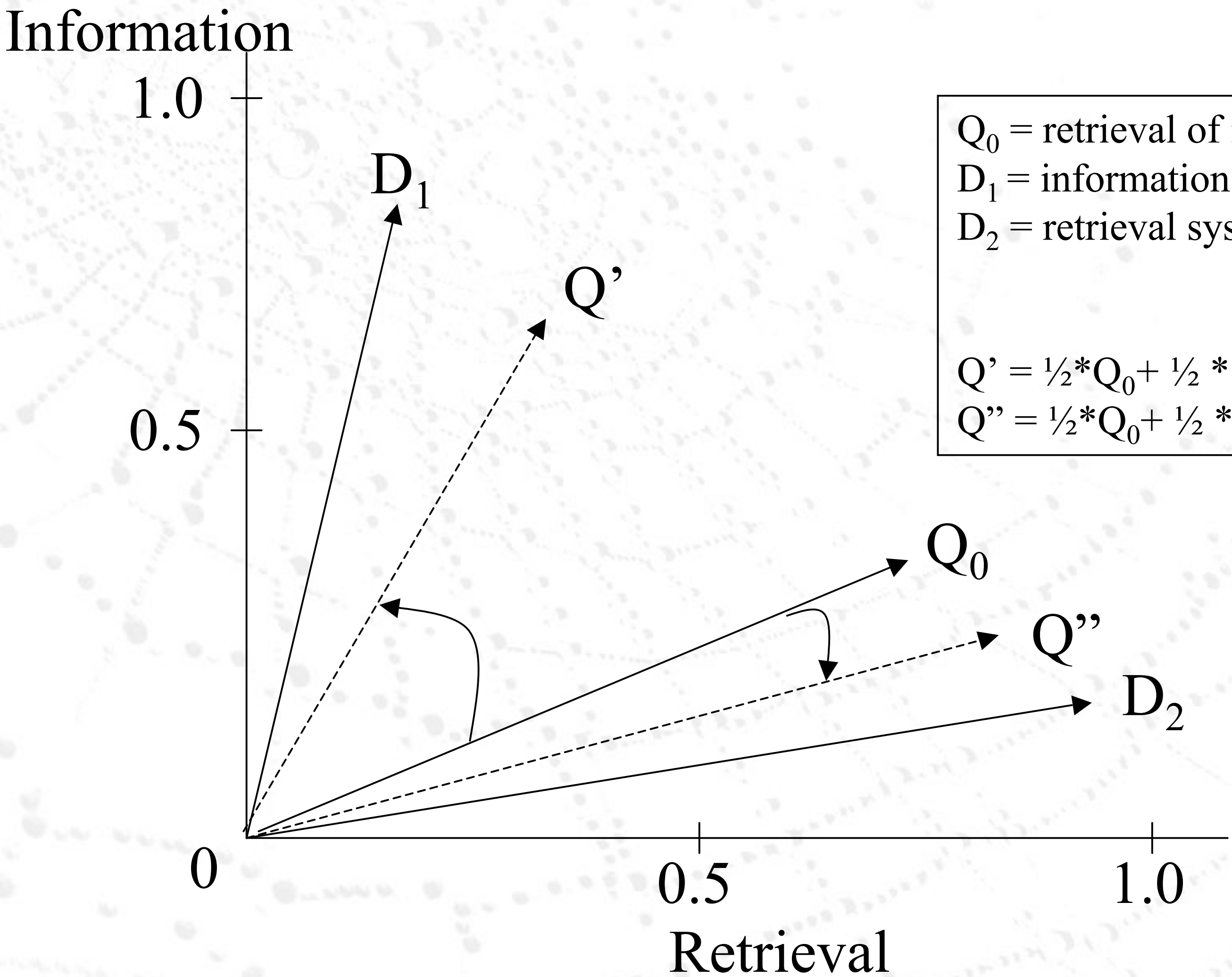
$n_1 =$ the number of relevant documents chosen

$n_2 =$ the number of non-relevant documents chosen

β and γ tune the importance of relevant and nonrelevant terms

(in some studies best to set β to 0.75 and γ to 0.25)

Rocchio/Vector Illustration



Q_0 = retrieval of information = (0.7,0.3)
D_1 = information science = (0.2,0.8)
D_2 = retrieval systems = (0.9,0.1)

$Q' = \frac{1}{2} * Q_0 + \frac{1}{2} * D_1 = (0.45,0.55)$
$Q'' = \frac{1}{2} * Q_0 + \frac{1}{2} * D_2 = (0.80,0.20)$

Evaluation Contingency Table

	System says is **relevant**	System says is **irrelevant**
Document is actually **relevant**	TP (True Positive)	FN (False Negative)
Document is actually **irrelevant**	FP (False Positive)	TN (True Negative)

Evaluation Metrics

$$\frac{TP}{TP+FP}$$

- Precision = Positive Predictive Value
  - "ratio of the number of relevant documents retrieved over the total number of documents retrieved"
  - how much extra stuff did you get?

$$\frac{TP}{TP+FN}$$

- Recall = Sensitivity
  - "ratio of relevant documents retrieved for a given query over the number of relevant documents for that query in the database"
  - how much did you miss?

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

- F_1 measure = harmonic mean of P and R
  - Can use other coefficients instead of 1

One number to rule them all: MAP

- A “standard” measure: Mean Average Precision (MAP)
  - average of precision at all points where a new relevant document is found.
  - Problem: favors systems with high   .
  - On the web, a user is usually looking just at the first a few results in Web search.
    - Leads to precision at *k* documents, but it’s kludgy: not sensitive to the ranking of every relevant document.

A second try: nDCG

- “**Gain**”: Each rel doc gives some level of relevance to the user
  G’ = <3,2,3,0,0,1>
- “**Cumulative**”: overall utility of *n* docs = sum of gain of each rel doc.
  CG’ = <3,5,8,8,8,9>
- “**Discount**” docs further down in list, as they are less likely to be used
  DCG’ = <3, 3+2/log2, 3+2/log2+3/log3, …, 3+2/log2+3/log3+1/log6>
- **“Normalized**” against ideal IR system rankings
  Ideal G’ = <3,3,2,1,0,0>
  Ideal DCG’ = <3, 3+3/log2, 3+3/log2+2/log3, 3+3/log2+2/log3+1/log4, …>
  nDCG’ = DCG’ / Ideal DCG’ = <1, …>

Pro: works naturally from fractional relevance
Con: have to set the discounting coefficients in NDCG (why log?)

To summarize

- TF - Favor terms important to the document
- IDF - Favor terms selective of the document
- Normalize documents of different length

- Docs and Queries all as vectors
  - Ask for help from the user to construct new query
  - Document as query - similarity search “more like this”
- Retrieval Evaluation as P/R/F_1 and nDGC.