CS276A

Information Retrieval

Lecture 6

Plan

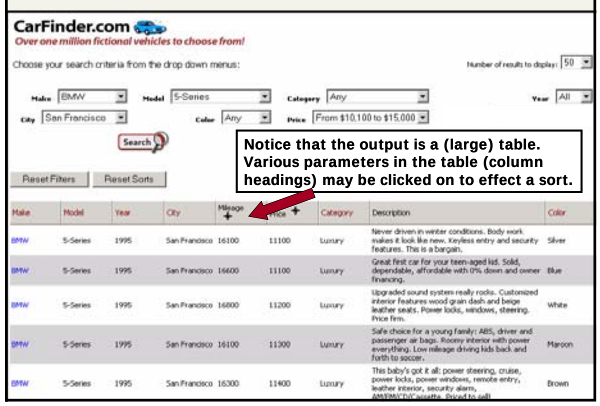
- Last lecture
  - Index construction
- This lecture
  - Parametric and field searches
    - Zones in documents
  - Scoring documents: zone weighting
    - Index support for scoring
  - Term weighting

Parametric search

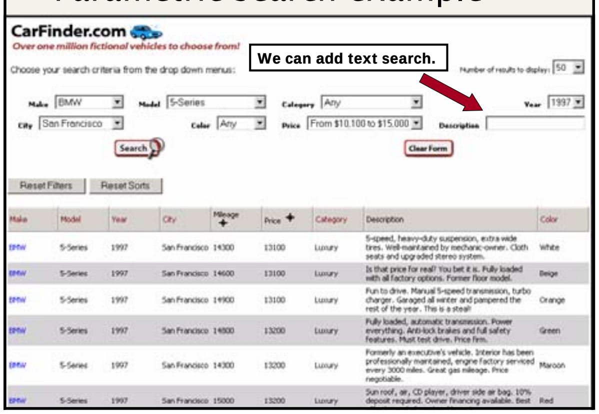
- Each document has, in addition to text, some "meta-data" in <u>fields</u> e.g.,
  - Language = French
  - Format = pdf
  - Subject = Physics etc.
  - Date = Feb 2000
- A parametric search interface allows the user to combine a full-text query with selections on these field values e.g.,
  - language,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

Fields → ← Values

Parametric search example



Parametric search example



Parametric/field search

- In these examples, we select field values
  - Values can be hierarchical, e.g.,
  - <u>Geography</u>: Continent → Country → State → City
- A paradigm for navigating through the document collection, e.g.,
  - "Aerospace companies in Brazil" can be arrived at first by selecting <u>Geography</u> then <u>Line of Business</u>, or vice versa
  - Filter docs in contention and run text searches scoped to subset

Index support for parametric search

- Must be able to support queries of the form
  - Find pdf documents that contain “stanford university”
  - A field selection (on doc format) and a phrase query
- Field selection – use inverted index of field values → docids
  - Organized by field name
  - Use compression etc. as before

Parametric index support

- Optional – provide richer search on field values – e.g., wildcards
  - Find books whose Author field contains ***s*trup***
- Range search – find docs authored between September and December
  - Inverted index doesn’t work (as well)
  - Use techniques from database range search
  - See for instance www.bluerwhite.org/btree/ for a summary of B-trees
- Use query optimization heuristics as before

Normalization

- For this to work, fielded data needs normalization
  - E.g., prices expressed variously as 13K, 28,500, $25,200, 28000
  - Simple grammars/rules normalize these into a single sort order

Field retrieval

- In some cases, must retrieve field values
  - E.g., ISBN numbers of books by ***s*trup***
- Maintain “forward” index – for each doc, those field values that are “retrievable”
  - Indexing control file specifies which fields are retrievable (and can be updated)
  - Storing primary data here, not just an index

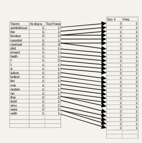
(as opposed to “inverted”)

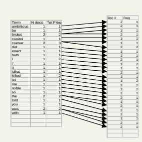
Zones

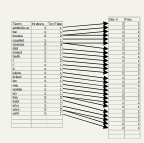
- A zone is an identified region within a doc
  - E.g., Title, Abstract, Bibliography
  - Generally culled from marked-up input or document metadata (e.g., powerpoint)
- Contents of a zone are free text
  - Not a “finite” vocabulary
- Indexes for each zone - allow queries like
  - ***sorting*** in Title AND ***smith*** in Bibliography AND ***recur**** in Body
- Not queries like “all papers whose authors cite themselves”

Why?

Zone indexes – simple view


Title


Author


Body

etc.

So we have a database now?

- Not really.
- Databases do lots of things we don't need
  - Transactions
  - Recovery (our index is not the system of record; if it breaks, simply reconstruct from the original source)
  - Indeed, we never have to store text in a search engine – only indexes
- We're focusing on optimized indexes for text-oriented queries, not a SQL engine.

Scoring

Scoring

- Thus far, our queries have all been Boolean
  - Docs either match or not
- Good for expert users with precise understanding of their needs and the corpus
- Applications can consume 1000's of results
- Not good for (the majority of) users with poor Boolean formulation of their needs
- Most users don't want to wade through 1000's of results – cf. altavista

Scoring

- *We wish to return in order the documents most likely to be useful to the searcher*
- How can we rank order the docs in the corpus with respect to a query?
- Assign a score – say in [0,1]
  - for each doc on each query
- Begin with a perfect world – no spammers
  - Nobody stuffing keywords into a doc to make it match queries
  - More on this in 276B under web search

Linear zone combinations

- First generation of scoring methods: use a linear combination of Booleans:
  - E.g.,

Score = 0.6*<***sorting*** in Title> + 0.3*<***sorting*** in Abstract> + 0.05*<***sorting*** in Body> + 0.05*<***sorting*** in Boldface>

  - Each expression such as <***sorting*** in Title> takes on a value in {0,1}.
  - Then the overall score is in [0,1].

For this example the scores can only take on a finite set of values – what are they?

Linear zone combinations

- In fact, the expressions between <> on the last slide could be *any* Boolean query
- Who generates the Score expression (with weights such as 0.6 etc.)?
  - In uncommon cases – the user through the UI
  - Most commonly, a query parser that takes the user's Boolean query and runs it on the indexes for each zone
  - Weights determined from user studies and hard-coded into the query parser.

Exercise

- On the query ***bill*** *OR* ***rights*** suppose that we retrieve the following docs from the various zone indexes:

Author	***bill***	1 → 2
	rights	
Title	***bill***	3 → 5 → 8
	rights	3 → 5 → 9
Body	***bill***	1 → 2 → 5 → 9
	rights	3 → 5 → 8 → 9

Compute the score for each doc based on the weightings 0.6,0.3,0.1

General idea

- We are given a weight vector whose components sum up to 1.
  - There is a weight for each zone/field.
- Given a Boolean query, we assign a score to each doc by adding up the weighted contributions of the zones/fields.
- Typically – users want to see the *K* highest-scoring docs.

Index support for zone combinations

- In the simplest version we have a separate inverted index for each zone
- Variant: have a single index with a separate dictionary entry for each term and zone
- E.g.,

bill.author	1 → 2
bill.title	3 → 5 → 8
bill.body	1 → 2 → 5 → 9

Of course, compress zone names like author/title/body.

Zone combinations index

- The above scheme is still wasteful: each term is potentially replicated for each zone
- In a slightly better scheme, we encode the zone in the postings:

bill 1.author, 1.body → 2.author, 2.body → 3.title

As before, the zone names get compressed.

- At query time, accumulate contributions to the total score of a document from the various postings, e.g.,

Score accumulation

1	0.7
2	0.7
3	0.4
5	0.4

bill 1.author, 1.body → 2.author, 2.body → 3.title

rights 3.title, 3.body → 5.title, 5.body →

- As we walk the postings for the query ***bill*** *OR* ***rights***, we accumulate scores for each doc in a linear merge as before.
- Note: we get both ***bill*** and ***rights*** in the Title field of doc 3, but score it no higher.
- Should we give more weight to more hits?

Free text queries

- Before we raise the score for more hits:
- We just scored the Boolean query ***bill*** *OR* ***rights***
- Most users more likely to type ***bill rights*** or ***bill of rights***
  - How do we interpret these “free text” queries?
  - No Boolean connectives
  - Of several query terms some may be missing in a doc
  - Only some query terms may occur in the title, etc.

Free text queries

- To use zone combinations for free text queries, we need
  - A way of assigning a score to a pair <free text query, zone>
  - Zero query terms in the zone should mean a zero score
  - More query terms in the zone should mean a higher score
  - Scores don't have to be Boolean
- Will look at some alternatives now

Incidence matrices

- Recall: Document (or a zone in it) is binary vector X in $\{0,1\}^v$
  - Query is a vector
- Score: Overlap measure:

$$|X \cap Y|$$

	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

Example

- On the query ***ides of march***, Shakespeare's *Julius Caesar* has a score of 3
- All other Shakespeare plays have a score of 2 (because they contain ***march***) or 1
- Thus in a rank order, *Julius Caesar* would come out tops

Overlap matching

- What's wrong with the overlap measure?
- It doesn't consider:
  - Term frequency in document
  - Term scarcity in collection (document mention frequency)
    - ***of*** is more common than ***ides*** or ***march***
  - Length of documents
    - (And queries: score not normalized)

Overlap matching

- One can normalize in various ways:
  - Jaccard coefficient:

$$|X \cap Y| / |X \cup Y|$$

  - Cosine measure:

$$|X \cap Y| / \sqrt{|X| \times |Y|}$$

- What documents would score best using Jaccard against a typical query?
  - Does the cosine measure fix this problem?

Scoring: density-based

- Thus far: position and overlap of terms in a doc – title, author etc.
- Obvious next idea: if a document talks about a topic *more,* then it is a better match
- This applies even when we only have a single query term.
- Document relevant if it has a lot of the terms
- This leads to the idea of term weighting.

Term weighting

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Bag of words model
  - Document is a 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 view of a doc

- Thus the doc
  - ***John is quicker than Mary***.

is indistinguishable from the doc

  - ***Mary is quicker than John***.

Which of the indexes discussed so far distinguish these two docs?

Counts vs. frequencies

- Consider again the ***ides of march*** query.
  - *Julius Caesar* has 5 occurrences of ***ides***
  - No other play has ***ides***
  - ***march*** occurs in over a dozen
  - All the plays contain ***of***
- By this scoring measure, the top-scoring play is likely to be the one with the most ***of***s

Digression: terminology

- WARNING: In a lot of IR literature, “frequency” is used to mean “count”
  - Thus *term frequency* in IR literature is used to mean *number of occurrences* in a doc
  - Not divided by document length (which would actually make it a frequency)
- We will conform to this misnomer
  - In saying term frequency we mean the number of occurrences of a term in a document.

Term frequency *tf*

- Long docs are favored because they’re more likely to contain query terms
- Can fix this to some extent by normalizing for document length
- But is raw *tf* the right measure?

Weighting term frequency: *tf*

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences …
- Unclear: while it seems that more is better, a lot isn't proportionally better than a few
  - Can just use raw *tf*
  - Another option commonly used in practice:

$$wf_{t,d} = 0 \text{ if } tf_{t,d} = 0,\ 1 + \log tf_{t,d} \text{ otherwise}$$

Score computation

- Score for a query *q* = sum over terms *t* in *q*:

$$= \sum_{t \in q} tf_{t,d}$$

- [Note: 0 if no query terms in document]
- This score can be zone-combined
- Can use *wf* instead of *tf* in the above
- Still doesn't consider term scarcity in collection (***ides*** is rarer than ***of***)

Weighting should depend on the term overall

- 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*
try	10422	8760
insurance	10440	3997

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

tf x idf term weights

- tf x idf measure combines:
  - term frequency (*tf*)
    - or *wf*, 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:

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

- See Kishore Papineni, NAACL 2, 2002 for theoretical justification

Summary: tf x idf (or tf.idf)

- Assign a tf.idf weight to each term *i* in each document *d*

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

What is the wt of a term that occurs in all of the docs?

$tf_{i,d}$ = frequency of term *i* in document *j*
n = total number of documents
df_i = the number of documents that contain term *i*

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus

Real-valued term-document matrices

- Function (scaling) of count of a word in a document:
  - Bag of words model
  - Each is a vector in $\mathbb{R}^v$
  - Here log-scaled *tf.idf*

Note can be >1!

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

Documents as vectors

- Each doc *j* can now be viewed as a vector of *wf*×*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
- (The corpus of documents gives us a matrix, which we could also view as a vector space in which words live – transposable data)

Recap

- We began by looking at zones at scoring
- Ended up viewing documents as vectors
- Will pursue this view next time.