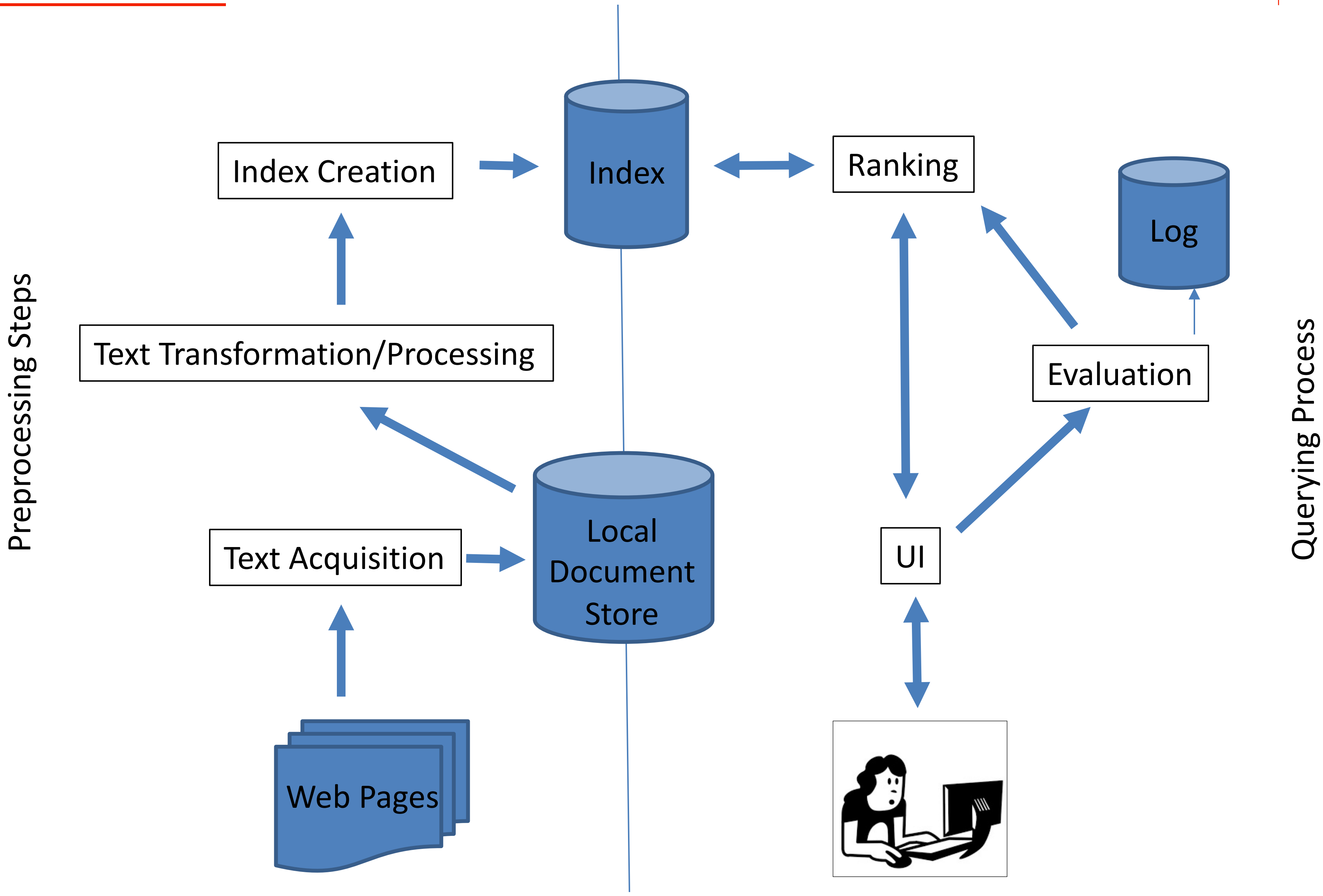
Informatics 225
Computer Science 221

Information Retrieval

Lecture 20

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. Powerpoint theme by Prof. André van der Hoek.

Architecture



Evolving from Boolean retrieval

Retrieval Models

- Provide a mathematical framework for defining the search process
  - includes explanation of assumptions
  - basis of ranking algorithms
  - can be implicit

Retrieval Models

- Provide a mathematical framework for defining the search process
  - includes explanation of assumptions
  - basis of ranking algorithms
  - can be implicit

- Progress in retrieval models has corresponded with improvements in effectiveness

Retrieval Models

- Provide a mathematical framework for defining the search process
  - includes explanation of assumptions
  - basis of ranking algorithms
  - can be implicit

- Progress in retrieval models has corresponded with improvements in effectiveness

- Theories about relevance

Relevance

- Complex concept that has been studied for some time
  - Many factors to consider
  - People often disagree when making relevance judgments

Relevance

- Complex concept that has been studied for some time
  - Many factors to consider
  - People often disagree when making relevance judgments

- Retrieval models make various assumptions about relevance to simplify problem
  - e.g., *topical* vs. *user* relevance
  - e.g., *binary* vs. *continuous* vs. *multi-valued* relevance

Ranked retrieval

- Thus far in INF225/CS221, our queries have all been Boolean.
  - Documents either match or don't.

Ranked retrieval

- Thus far in INF225/CS221, 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.

Ranked retrieval

- Thus far in INF225/CS221, 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.
- Not good for the majority of "generic" users.
  - Most users incapable of writing Boolean queries; *or they are capable, but they think it's too much work…*
  - Most users don't want to wade through 1000s of results.
    - *Particularly true for web search.*

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.

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

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 some skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many.
  - Complex combinations are necessary, and except in specific cases (e.g. law), people happily trade deterministic results by ease of use.

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.

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.

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.
- In principle, there are two separate choices here, but in practice, ranked retrieval has 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 the top k (≈ 10) results, or show them in the first “page”
  - We don’t overwhelm the user

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 the top *k* (≈ 10) results, or show them in the first "page"
  - We don't overwhelm the user
  - Strong assumption: the ranking algorithm "works" for the user

Scoring as the basis of ranked retrieval

- We wish to return the documents in an order that is most likely to be useful to the searcher

Scoring as the basis of ranked retrieval

- We wish to return the documents in an order that is 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

- We need a way of assigning a score to a query/document pair

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a simple one-term query

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a simple one-term query
  - If the query term does not occur in the document: score should be 0

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a simple 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)

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a simple 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 some alternatives for this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*
- jaccard$(A,B) = |A \cap B| \;/\; |A \cup B|$

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*
- jaccard$(A,B) = |A \cap B| \,/\, |A \cup B|$
- jaccard$(A,A) = 1$
- jaccard$(A,B) = 0$ if $A \cap B = 0$

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*
- jaccard$(A,B) = |A \cap B| \,/\, |A \cup B|$
- jaccard$(A,A) = 1$
- jaccard$(A,B) = 0$ if $A \cap B = 0$
- Good properties:
  - *A* and *B* don't have to be the same size.
  - Always assigns a number between 0 and 1.

Issues with Jaccard for scoring

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

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.

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.
- More sophisticated heuristics to normalize for the length of the documents result in better scores

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.
- More sophisticated heuristics to normalize for the length of the documents result in better scores
- Another option: $|A \cap B| / \sqrt{|A \cup B|}$
  instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization.

Recall the 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

Bag of words model

- Vector representation doesn't consider the ordering of words in a document

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**

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.

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 temporary step back, since a positional index is able to distinguish these two documents.
  - We will see positional information later
  - For now: bag of words model

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*.

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*.

- We want to use tf when computing query-document match scores. But how?

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*.

- 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.

Term frequency tf

- The term frequency $tf_{t,d}$ 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.

Note: frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

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

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

Log-frequency weighting

- 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, 2 → 1.3, 10 → 2, 1000 → 4, etc.

- Score for a document-query pair: sum over terms *t* in both *q* and *d.*

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

- 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

Document frequency

- **Rare terms are more informative than frequent terms**
  - Recall stop words
- Consider a term in the query that is rare in the collection
  - e.g., *arachnocentric*

Document frequency

- **Rare terms are more informative than frequent terms**
  - Recall stop words
- Consider a term in the query that is rare in the collection
  - e.g., *arachnocentric*
- A document containing this term is very likely to be relevant to the query *arachnocentric.*

Document frequency

- **Rare terms are more informative than frequent terms**
  – Recall stop words
- Consider a term in the query that is rare in the collection
  – e.g., *arachnocentric*
- A document containing this term is very likely to be relevant to the query *arachnocentric.*

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

We can do better than simple term frequencies…

Document frequency, continued

- Frequent terms in the collection are less informative than rare terms

- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)

- A document containing such a term is **more likely to be relevant** than a document that doesn't

- **But it's not a sure indicator of relevance.**

→ 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
  - $0 < \mathrm{df}_t \leq N$

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
  - $0 < \text{df}_t \leq N$
- 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* = 1 million

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

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

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

Effect of idf on ranking

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

Effect of idf on ranking

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

- 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.

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 thus 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.

$$w_{t,d} = (1+\log(tf_{t,d})) \times \log(N/df_t)$$

Relevance or Score for a document *d* given a query *q*

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

tf-idf weighting

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

$$w_{t,d} = (1+\log(tf_{t,d})) \times \log(N/df_t)$$

- Best known and most used weighting scheme in IR
  - **Note: the "-" in tf-idf is a hyphen, not a minus sign!**
  - Alternative names: tf.idf, tf x idf, tfidf

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

Relevance or Score for a document *d* given a query *q*

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

- Many variants can be constructed
  - E.g. How “tf” is computed (e.g. with/without logs)
  - E.g. Whether the terms in the query are also weighted
  - … *pick and/or design your heuristics…*