

Informatics 225

Computer Science 221

Information Retrieval

Lecture 19

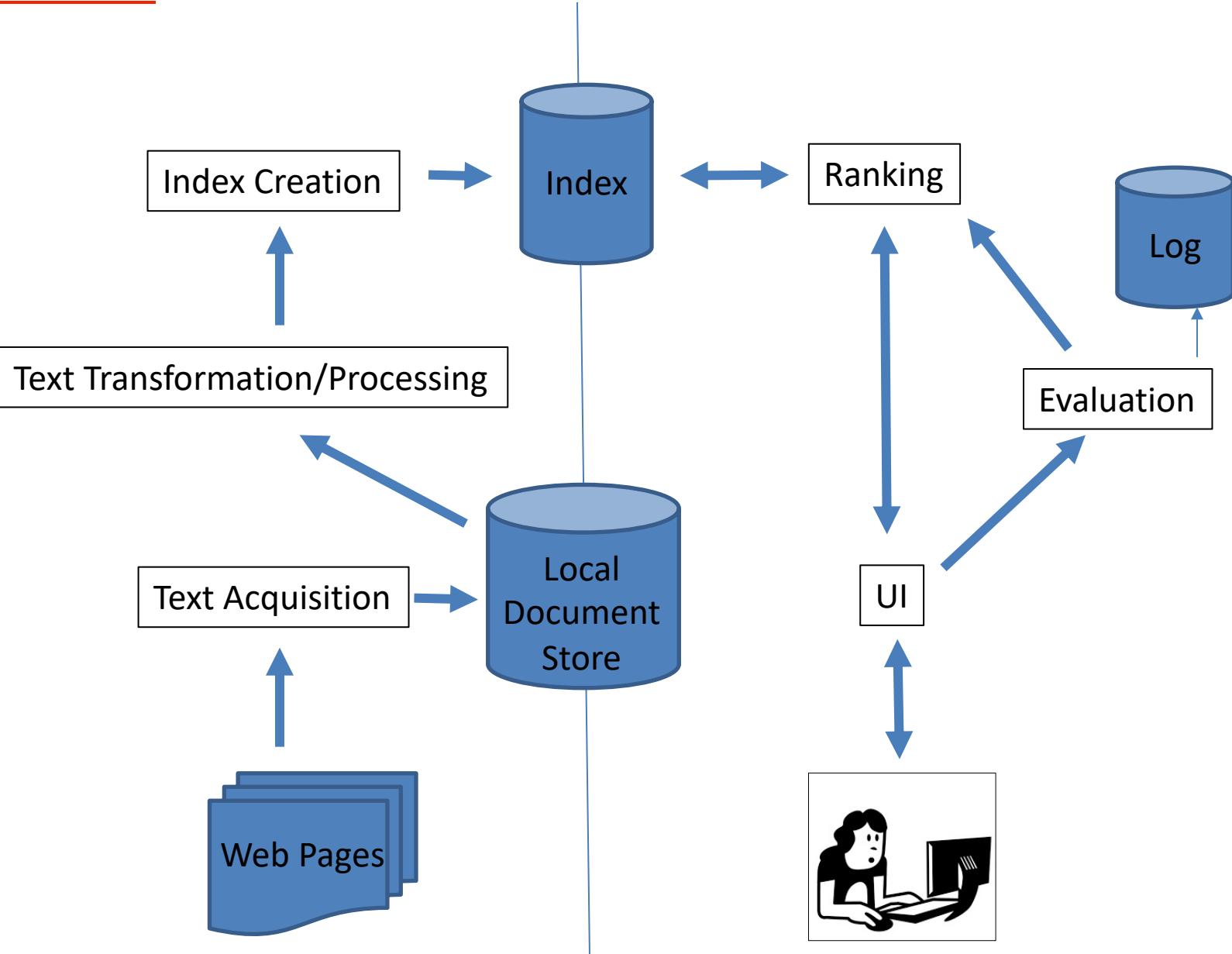
*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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.

Optimizing Query Evaluation

Architecture

Preprocessing Steps



Distributed Query Evaluation

- Will speed up significantly the query processing speed

Distributed Query Evaluation

- Will speed up significantly the query processing speed
- Basic process
 - All queries sent to a *director machine*

Distributed Query Evaluation



- Will speed up significantly the query processing speed
- Basic process
 - All queries sent to a *director machine*
 - Director then sends messages to many *index servers*

Distributed Query Evaluation

- Will speed up significantly the query processing speed
- Basic process
 - All queries sent to a *director machine*
 - Director then sends messages to many *index servers*
 - Each index server does some portion of the query processing

Distributed Query Evaluation



- Will speed up significantly the query processing speed
- Basic process
 - All queries sent to a *director machine*
 - Director then sends messages to many *index servers*
 - Each index server does some portion of the query processing
 - Director organizes the results and returns them to the user

Distributed Query Evaluation



- Will speed up significantly the query processing speed
- Basic process
 - All queries sent to a *director machine*
 - Director then sends messages to many *index servers*
 - Each index server does some portion of the query processing
 - Director organizes the results and returns them to the user
- Two main approaches on how to distribute the query
 - Document distribution
 - Term distribution

Distributed Query Evaluation



- Document distribution
 - each index server acts as a search engine for a small fraction of the total collection

Distributed Query Evaluation



- Document distribution
 - each index server acts as a search engine for a small fraction of the total collection
 - director sends a copy of the query to each of the index servers, each of which returns the top- k results

Distributed Query Evaluation



- Document distribution
 - each index server acts as a search engine for a small fraction of the total collection
 - director sends a copy of the query to each of the index servers, each of which returns the top- k results
 - results are merged into a single ranked list by the director

Distributed Query Evaluation



- Document distribution
 - each index server acts as a search engine for a small fraction of the total collection
 - director sends a copy of the query to each of the index servers, each of which returns the top- k results
 - results are merged into a single ranked list by the director
- Collection statistics should be shared for effective ranking

Distributed Query Evaluation

- Term distribution
 - Single index is built for the whole cluster of machines

Distributed Query Evaluation



- Term distribution
 - Single index is built for the whole cluster of machines
 - Each inverted list in that index is then assigned to one index server
 - in most cases the data to process a query is not stored on a single machine

Distributed Query Evaluation



- Term distribution
 - Single index is built for the whole cluster of machines
 - Each inverted list in that index is then assigned to one index server
 - in most cases the data to process a query is not stored on a single machine
 - One of the index servers is chosen to process the query
 - usually the one holding the longest inverted list

Distributed Query Evaluation

- Term distribution
 - Single index is built for the whole cluster of machines
 - Each inverted list in that index is then assigned to one index server
 - in most cases the data to process a query is not stored on a single machine
 - One of the index servers is chosen to process the query
 - usually the one holding the longest inverted list
 - Other index servers send information to that server

Distributed Query Evaluation

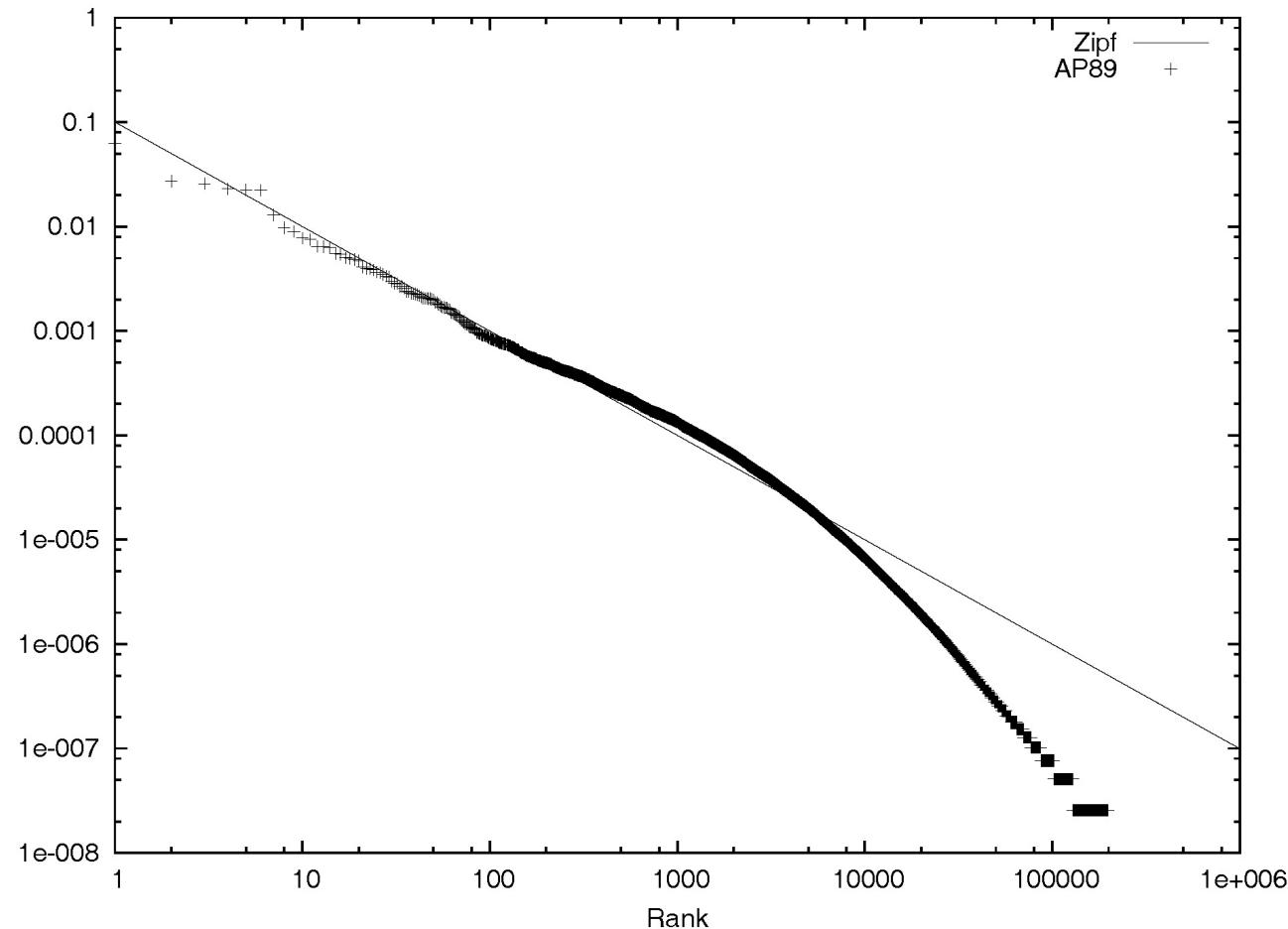
- Term distribution
 - Single index is built for the whole cluster of machines
 - Each inverted list in that index is then assigned to one index server
 - in most cases the data to process a query is not stored on a single machine
 - One of the index servers is chosen to process the query
 - usually the one holding the longest inverted list
 - Other index servers send information to that server
 - Final results sent to director



Query Caching

- **Query distributions similar to Zipf**
 - About $\frac{1}{2}$ each day are unique, but some are very popular

Reminder: Zipf's Law for AP89



- Note problems at high and low frequencies
- Words that occur once : *Hapax Legomena*.

Query Caching



- **Query distributions similar to Zipf**
 - About $\frac{1}{2}$ each day are unique, but some are very popular

Query Caching



- Query distributions similar to Zipf
 - About $\frac{1}{2}$ each day are unique, but some are very popular
- **Caching can significantly improve effectiveness**
 - Cache (perhaps in memory) popular query results
 - Cache in memory common inverted lists

Query Caching



- Query distributions similar to Zipf
 - About $\frac{1}{2}$ each day are unique, but some are very popular
- Caching can significantly improve effectiveness
 - Cache (perhaps in memory) popular query results
 - Cache in memory common inverted lists
- Inverted list caching can help even with unique queries

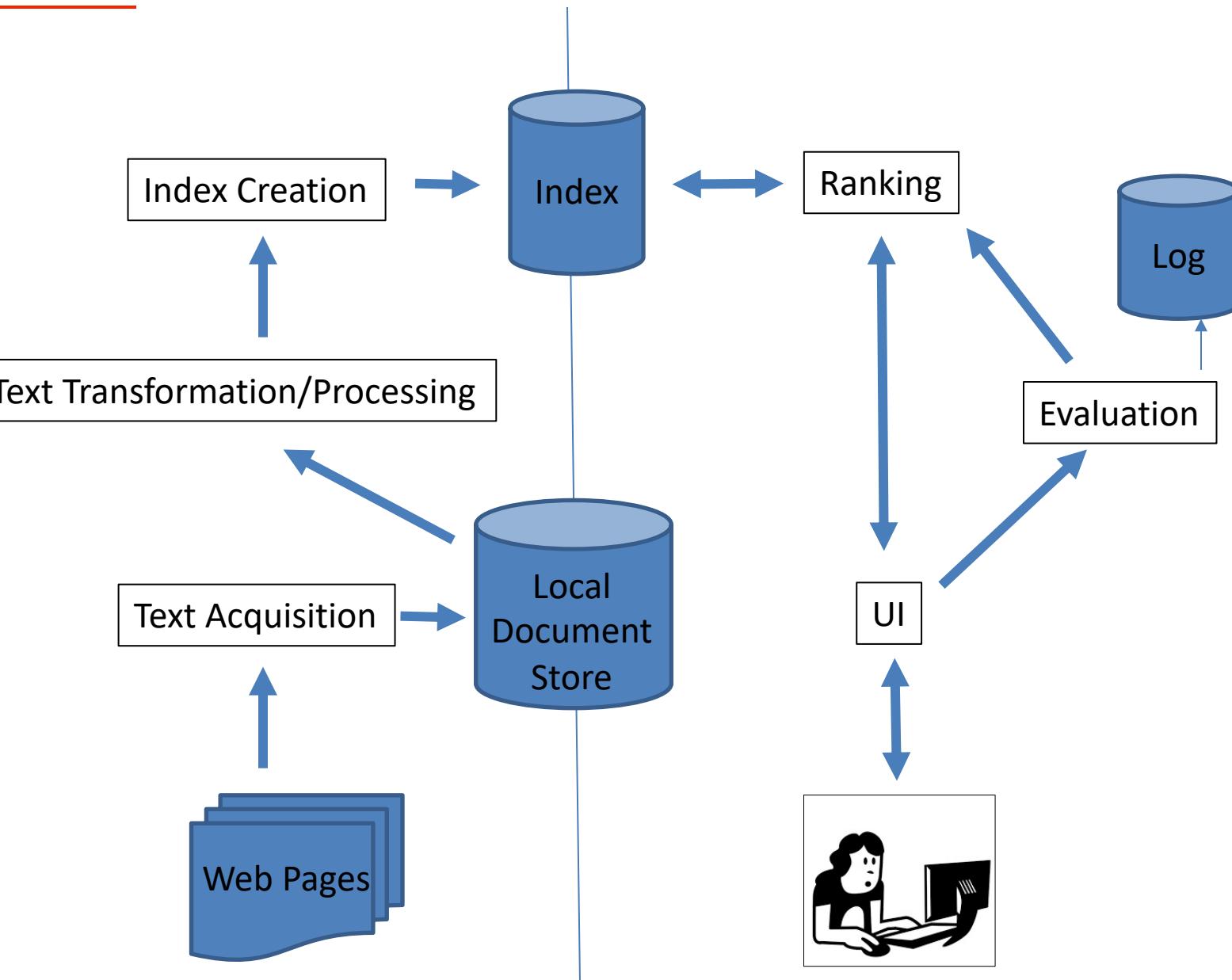
Query Caching



- Query distributions similar to Zipf
 - About $\frac{1}{2}$ each day are unique, but some are very popular
- Caching can significantly improve effectiveness
 - Cache (perhaps in memory) popular query results
 - Cache in memory common inverted lists
- Inverted list caching can help even with unique queries
- Cache must be refreshed to prevent stale data

Architecture

Preprocessing Steps



Query Processing



- Document-at-a-time
- Term-at-a-time

Query Processing



- **Document-at-a-time**
 - Calculates complete scores for documents by processing all term lists, one document at a time

Query Processing



- Document-at-a-time
 - Calculates complete scores for documents by processing all term lists, one document at a time
- Term-at-a-time
 - Accumulates scores for documents by processing term lists one at a time

Query Processing



- Document-at-a-time
 - Calculates complete scores for documents by processing all term lists, one document at a time
- Term-at-a-time
 - Accumulates scores for documents by processing term lists one at a time
- Both approaches have optimization techniques that significantly reduce time required to generate scores

Document-At-A-Time



Query : salt water tropical Postings (x:y) – document number and word count

salt	1:1
water	1:1
tropical	1:2
score	1:4

Document-At-A-Time

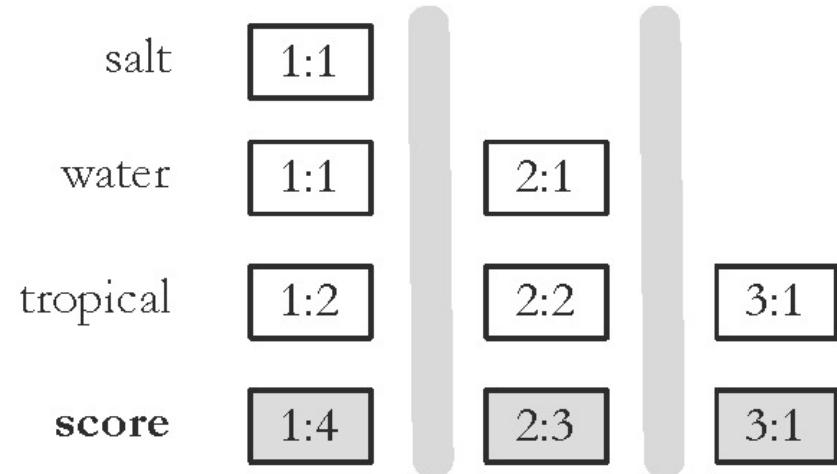
Query : salt water tropical Postings (x:y) – document number and word count

salt	1:1	
water	1:1	2:1
tropical	1:2	2:2
score	1:4	2:3

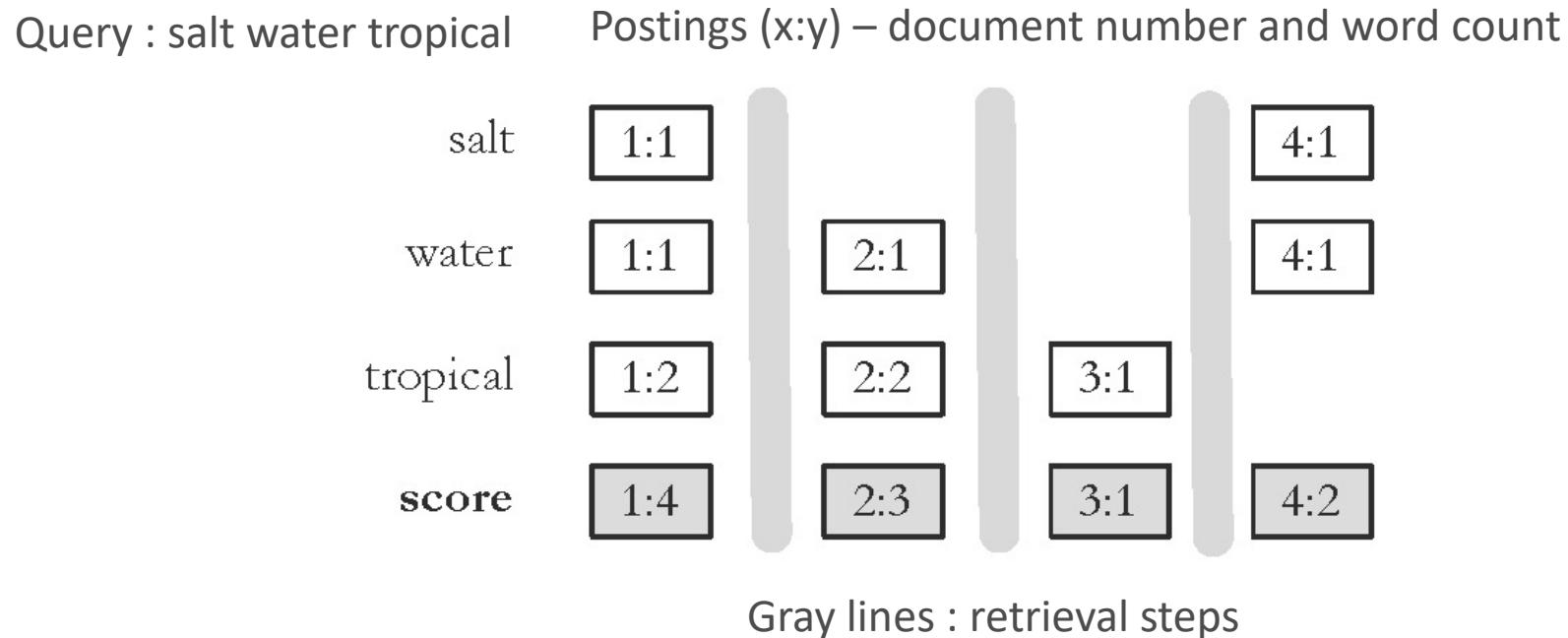
Gray lines : retrieval steps

Document-At-A-Time

Query : salt water tropical Postings (x:y) – document number and word count



Document-At-A-Time



Document-At-A-Time

```
procedure DOCUMENTATATIMERETRIEVAL( $Q, I, f, g, k$ )
     $L \leftarrow \text{Array}()$ 
     $R \leftarrow \text{PriorityQueue}(k)$ 
    for all terms  $w_i$  in  $Q$  do
         $l_i \leftarrow \text{InvertedList}(w_i, I)$ 
         $L.\text{add}( l_i )$ 
    end for
    for all documents  $d \in I$  do
         $s_d \leftarrow 0$ 
        for all inverted lists  $l_i$  in  $L$  do
            if  $l_i.\text{getCurrentDocument}() = d$  then
                 $s_d \leftarrow s_d + g_i(Q)f_i(l_i)$             $\triangleright$  Update the document score
            end if
             $l_i.\text{movePastDocument}( d )$ 
        end for
         $R.\text{add}( s_d, d )$ 
    end for
    return the top  $k$  results from  $R$ 
end procedure
```

Pseudocode Function Descriptions

- `getCurrentDocument()`
 - Returns the document number of the current posting of the inverted list.
- `skipForwardToDocument(d)`
 - Moves forward in the inverted list until `getCurrentDocument() <= d`. This function may read to the end of the list.
- `movePastDocument(d)`
 - Moves forward in the inverted list until `getCurrentDocument() < d`.
- `moveToNextDocument()`
 - Moves to the next document in the list. Equivalent to `movePastDocument(getCurrentDocument())`.
- `getNextAccumulator(d)`
 - returns the first document number $d' \geq d$ that has already has an accumulator.
- `removeAccumulatorsBetween(a, b)`
 - Removes all accumulators for documents numbers between a and b. A_d will be removed iff $a < d < b$.

Term-At-A-Time



Query : salt water tropical

Postings (x:y) – document number and word count

salt

1:1	4:1
-----	-----

partial scores

1:1	4:1
-----	-----

Gray lines :
retrieval steps

Term-At-A-Time

Query : salt water tropical

Postings (x:y) – document number and word count

salt

1:1	4:1
-----	-----

partial scores

1:1	4:1
-----	-----

old partial scores

1:1	4:1
-----	-----

water

1:1	2:1	4:1
-----	-----	-----

new partial scores

1:2	2:1	4:2
-----	-----	-----

Gray lines :
retrieval steps

Term-At-A-Time

Query : salt water tropical

Postings (x:y) – document number and word count

salt

1:1	4:1
-----	-----

partial scores

1:1	4:1
-----	-----

old partial scores

1:1	4:1
-----	-----

water

1:1	2:1	4:1
-----	-----	-----

new partial scores

1:2	2:1	4:2
-----	-----	-----

Gray lines :
retrieval steps

old partial scores

1:2	2:1	4:2
-----	-----	-----

tropical

1:2	2:2	3:1
-----	-----	-----

final scores

1:4	2:3	2:2	4:2
-----	-----	-----	-----

Term-At-A-Time

```
procedure TERMATATIMERETRIEVAL( $Q, I, f, g, k$ )
     $A \leftarrow \text{HashTable}()$ 
     $L \leftarrow \text{Array}()$ 
     $R \leftarrow \text{PriorityQueue}(k)$ 
    for all terms  $w_i$  in  $Q$  do
         $l_i \leftarrow \text{InvertedList}(w_i, I)$ 
         $L.\text{add}( l_i )$ 
    end for
    for all lists  $l_i \in L$  do
        while  $l_i$  is not finished do
             $d \leftarrow l_i.\text{getCurrentDocument}()$ 
             $A_d \leftarrow A_d + g_i(Q)f(l_i)$ 
             $l_i.\text{moveToNextDocument}()$ 
        end while
    end for
    for all accumulators  $A_d$  in  $A$  do
         $s_d \leftarrow A_d$                                  $\triangleright$  Accumulator contains the document score
         $R.\text{add}( s_d, d )$ 
    end for
    return the top  $k$  results from  $R$ 
end procedure
```

Optimization Techniques



- Term-at-a-time uses more memory for accumulators, but accesses disk more efficiently

Optimization Techniques

- Term-at-a-time uses more memory for accumulators, but accesses disk more efficiently
- Two classes of optimization
 - Read less data from inverted lists
 - e.g., skip lists (already covered on previous lectures. e.g. skip pointers), **index the index**
 - better for simple feature functions

Optimization Techniques

- Term-at-a-time uses more memory for accumulators, but accesses disk more efficiently
- Two classes of optimization
 - Read less data from inverted lists
 - e.g., skip lists (already covered on previous lectures. e.g. skip pointers), index the index
 - better for simple feature functions
 - Calculate scores for fewer documents
 - E.g., conjunctive processing
 - better for complex feature functions

Conjunctive processing



- Among the simplest types of query optimization.
- Default mode of many engines.

-

Conjunctive processing



- Among the simplest types of query optimization.
- Default mode of many engines.
- Every returned document must contain all query terms.

-

Conjunctive processing



- Among the simplest types of query optimization.
- Default mode of many engines.
- Every returned document must contain all query terms.
- **Works best when one of the query terms is rare.**
 - By first selecting the documents containing the rare term(s), you can skip a large fraction of the documents

Conjunctive processing



- Among the simplest types of query optimization.
- Default mode of many engines.
- Every returned document must contain all query terms.
- Works best when one of the query terms is rare.
 - By first selecting the documents containing the rare term(s), you can skip a large fraction of the documents
- **Can be used in Term-at-a-time and Document-at-a-time approaches**

Conjunctive Term-at-a-Time

```

1: procedure TERMATATIMERETRIEVAL( $Q, I, f, g, k$ )
2:    $A \leftarrow \text{Map}()$ 
3:    $L \leftarrow \text{Array}()$ 
4:    $R \leftarrow \text{PriorityQueue}(k)$ 
5:   for all terms  $w_i$  in  $Q$  do
6:      $l_i \leftarrow \text{InvertedList}(w_i, I)$ 
7:      $L.\text{add}( l_i )$ 
8:   end for
9:   for all lists  $l_i \in L$  do
10:     $d_0 \leftarrow -1$ 
11:    while  $l_i$  is not finished do
12:      if  $i = 0$  then
13:         $d \leftarrow l_i.\text{getCurrentDocument}()$ 
14:         $A_d \leftarrow A_d + g_i(Q)f(l_i)$ 
15:         $l_i.\text{moveToNextDocument}()$ 
16:      else
17:         $d \leftarrow l_i.\text{getCurrentDocument}()$ 
18:         $d' \leftarrow A.\text{getNextAccumulator}(d)$ 
19:         $A.\text{removeAccumulatorsBetween}(d_0, d')$ 
20:        if  $d = d'$  then
21:           $A_d \leftarrow A_d + g_i(Q)f(l_i)$ 
22:           $l_i.\text{moveToNextDocument}()$ 
23:        else
24:           $l_i.\text{skipForwardToDocument}(d')$ 
25:        end if
26:         $d_0 \leftarrow d'$ 
27:      end if
28:    end while
29:  end for
30:  for all accumulators  $A_d$  in  $A$  do
31:     $s_d \leftarrow A_d$             $\triangleright$  Accumulator contains the document score
32:     $R.\text{add}( s_d, d )$ 
33:  end for
34:  return the top  $k$  results from  $R$ 
35: end procedure

```

Check the accumulator for the next document containing all previously seen terms and skip the list of postings to that document.

```

1: procedure DOCUMENTATATIMEREtrieval( $Q, I, f, g, k$ )
2:    $L \leftarrow \text{Array}()$ 
3:    $R \leftarrow \text{PriorityQueue}(k)$ 
4:   for all terms  $w_i$  in  $Q$  do
5:      $l_i \leftarrow \text{InvertedList}(w_i, I)$ 
6:      $L.\text{add}( l_i )$ 
7:   end for
8:    $d \leftarrow -1$ 
9:   while all lists in  $L$  are not finished do
10:     $s_d \leftarrow 0$ 
11:    for all inverted lists  $l_i$  in  $L$  do
12:      if  $l_i.\text{getCurrentDocument}() > d$  then
13:         $d \leftarrow l_i.\text{getCurrentDocument}()$ 
14:      end if
15:    end for
16:    for all inverted lists  $l_i$  in  $L$  do
17:       $l_i.\text{skipForwardToDocument}(d)$ 
18:      if  $l_i.\text{getCurrentDocument}() = d$  then
19:         $s_d \leftarrow s_d + g_i(Q)f_i(l_i)$             $\triangleright$  Update the document score
20:         $l_i.\text{movePastDocument}( d )$ 
21:      else
22:         $d \leftarrow -1$ 
23:        break
24:      end if
25:    end for
26:    if  $d > -1$  then  $R.\text{add}( s_d, d )$ 
27:    end if
28:  end while
29:  return the top  $k$  results from  $R$ 
30: end procedure

```

Conjunctive Document-at-a-Time

Finds the largest document pointed by an inverted list

Try to skip all lists to point to the document

If it can, score, otherwise break.

Threshold Methods

- Threshold methods use number of top-ranked documents needed (k) to optimize query processing
 - for most applications, k is small

Threshold Methods



- Threshold methods use number of top-ranked documents needed (k) to optimize query processing
 - for most applications, k is small
- For any query, there is a *minimum score* that each document needs to reach before it can be shown to the user

Threshold Methods



- Threshold methods use number of top-ranked documents needed (k) to optimize query processing
 - for most applications, k is small
- For any query, there is a *minimum score* that each document needs to reach before it can be shown to the user
 - score of the k th-highest scoring document
 - gives *threshold* τ
 - Unfortunately, we don't know τ ...
 - **optimization methods estimate τ' to ignore documents**

Threshold Methods

- For document-at-a-time processing, use score of lowest-ranked document so far for τ'

Threshold Methods



- For document-at-a-time processing, use score of lowest-ranked document so far for τ'
 - for term-at-a-time, have to use k_{th} -largest score in the accumulator table

Threshold Methods



- For document-at-a-time processing, use score of lowest-ranked document so far for τ'
 - for term-at-a-time, have to use k_{th} -largest score in the accumulator table
- After you have some estimate:
 - *MaxScore* method compares the maximum score that remaining documents could have to τ'

Threshold Methods



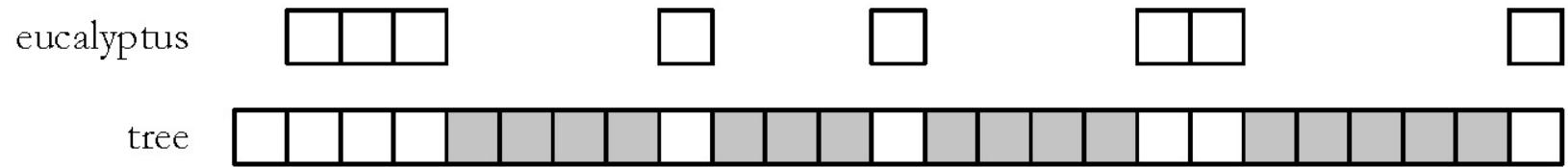
- For document-at-a-time processing, use score of lowest-ranked document so far for τ'
 - for term-at-a-time, have to use k_{th} -largest score in the accumulator table
- After you have some estimate:
 - *MaxScore* method compares the maximum score that remaining documents could have to τ'
 - Ignore parts of the inverted lists that will not generate document scores above τ'

Threshold Methods



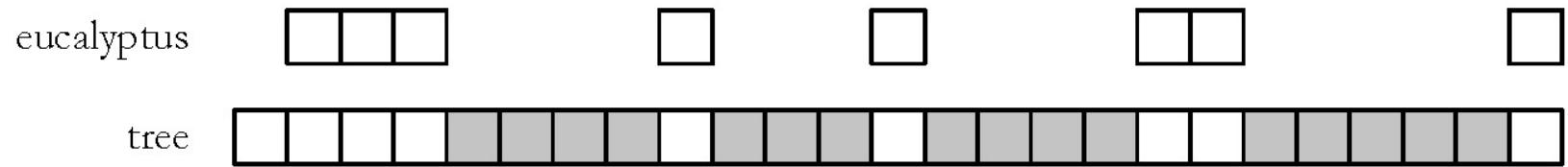
- For document-at-a-time processing, use score of lowest-ranked document so far for τ'
 - for term-at-a-time, have to use k_{th} -largest score in the accumulator table
- After you have some estimate:
 - *MaxScore* method compares the maximum score that remaining documents could have to τ'
 - Ignore parts of the inverted lists that will not generate document scores above τ'
 - *safe* optimization in that ranking will be the same without optimization

MaxScore Example



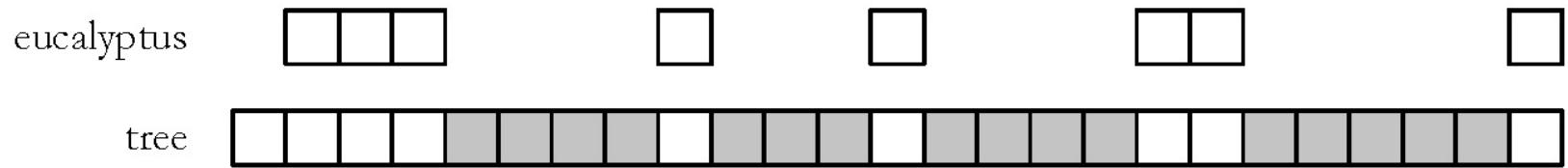
- Suppose that Indexer computes μ_{tree}
 - maximum score for any document containing just “tree”

MaxScore Example



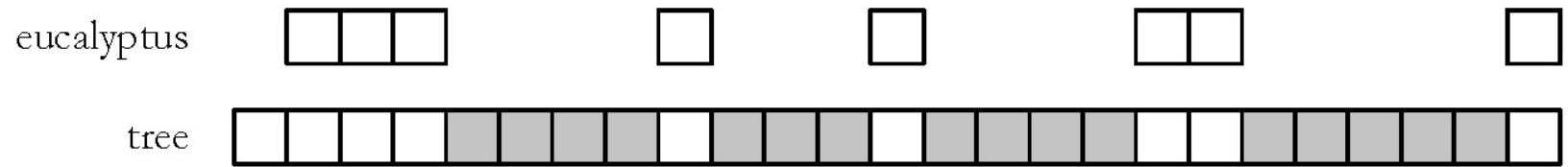
- Suppose that Indexer computes μ_{tree}
 - maximum score for any document containing just “tree”
- Assume $k = 3$, τ' is lowest score after first three docs

MaxScore Example



- Suppose that Indexer computes μ_{tree}
 - maximum score for any document containing just “tree”
- Assume $k = 3$, τ' is lowest score after first three docs
- Likely that $\tau' > \mu_{tree}$
 - τ' is the score of a document that contains **both query terms**

MaxScore Example



- Suppose that Indexer computes μ_{tree}
 - maximum score for any document containing just “tree”
- Assume $k = 3$, τ' is lowest score after first three docs
- Likely that $\tau' > \mu_{tree}$
 - τ' is the score of a document that contains both query terms
- **Can safely skip over all gray postings**

Other Approaches



- Early termination of query processing
 - Some queries are more expensive to compute than others...

Other Approaches



- Early termination of query processing
 - Some queries are more expensive to compute than others...
 - ignore high-frequency word lists in term-at-a-time

Other Approaches



- **Early termination of query processing**
 - Some queries are more expensive to compute than others...
 - ignore high-frequency word lists in term-at-a-time
 - ignore documents at end of lists in doc-at-a-time

Other Approaches



- Early termination of query processing
 - Some queries are more expensive to compute than others...
 - ignore high-frequency word lists in term-at-a-time
 - ignore documents at end of lists in doc-at-a-time
 - *unsafe* optimization

Other Approaches



- Early termination of query processing
 - Some queries are more expensive to compute than others...
 - ignore high-frequency word lists in term-at-a-time
 - ignore documents at end of lists in doc-at-a-time
 - *unsafe* optimization
- List ordering
 - order inverted lists by quality metric (e.g. PageRank) or by partial score

Other Approaches



- Early termination of query processing
 - Some queries are more expensive to compute than others...
 - ignore high-frequency word lists in term-at-a-time
 - ignore documents at end of lists in doc-at-a-time
 - *unsafe* optimization
- List ordering
 - order inverted lists by quality metric (e.g. PageRank) or by partial score
 - makes unsafe (and fast) optimizations more likely to retrieve good documents

Informatics 225

Computer Science 221

Information Retrieval

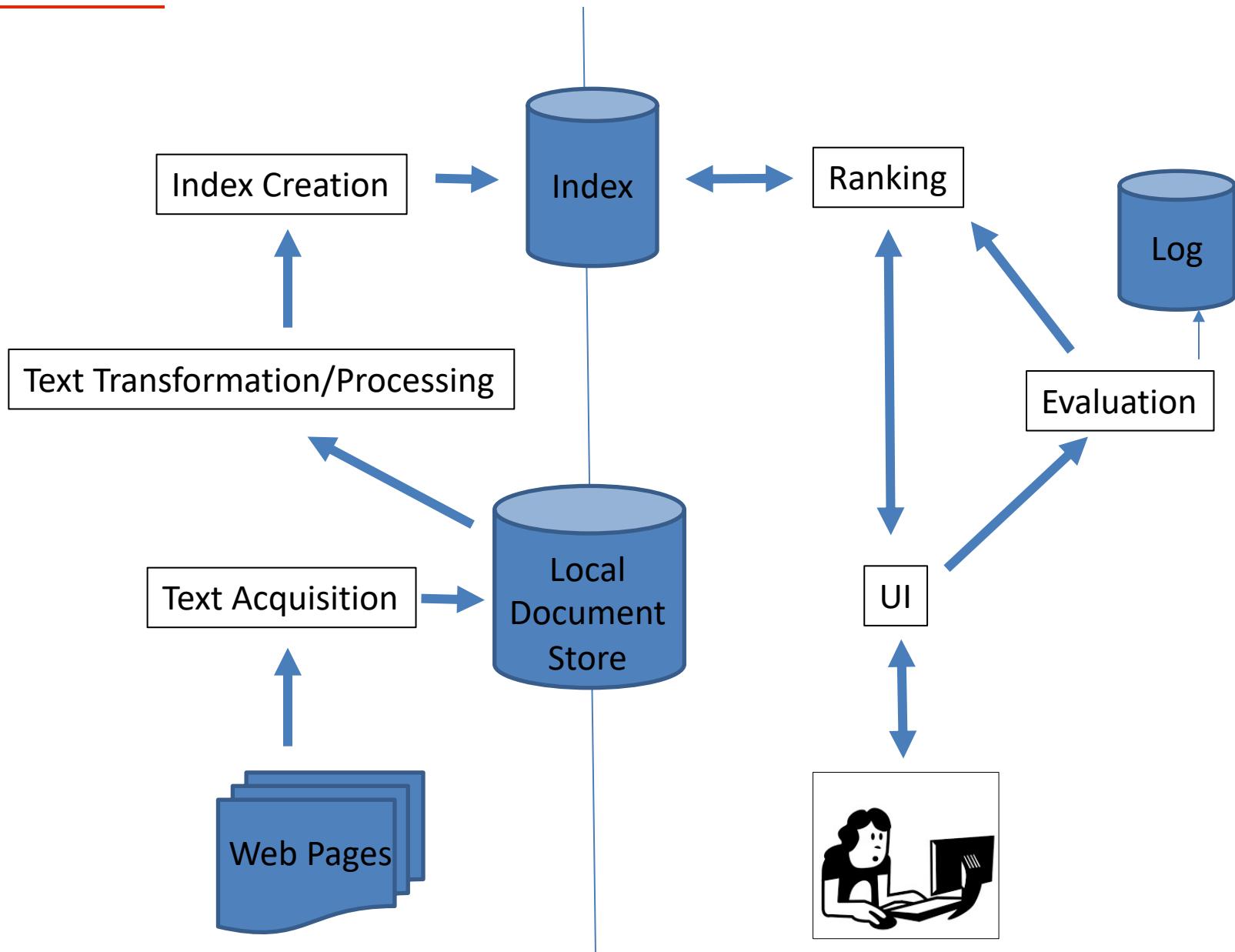
Lecture 20

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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

Preprocessing Steps



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



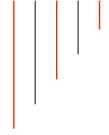
- Complex concept that has been studied for some time
 - Many factors to consider
 - People often disagree when making relevance judgments
- Retrieval models make various assumptions about relevance to simplify problem
 - e.g., *topical* vs. *user* relevance
 - e.g., *binary* vs. *continuous* vs. *multi-valued* relevance

Ranked retrieval



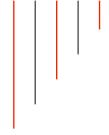
- Thus far in INF225/CS221, our queries have all been **Boolean**.
 - Documents either match or don't.

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



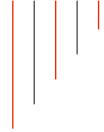
- Rather than a set of documents satisfying a query expression, in **ranked retrieval**, the system returns an ordering over the (top) documents in the collection for a query.

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

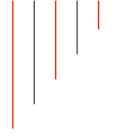


- We wish to return the documents in an order that is most likely to be useful to the searcher

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

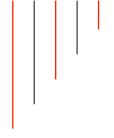


- A commonly used measure of overlap of two sets A and B

Take 1: Jaccard coefficient

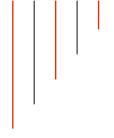
- A commonly used measure of overlap of two sets A and B
- $\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$

Take 1: Jaccard coefficient



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

Take 1: Jaccard coefficient



- A commonly used measure of overlap of two sets A and B
- $\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$
- $\text{jaccard}(A,A) = 1$
- $\text{jaccard}(A,B) = 0$ 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 $\text{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d .

Term frequency tf

- The term frequency $\text{tf}_{t,d}$ 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?

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.

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

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

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.

Log-frequency weighting

- The log frequency weight of term t in d is

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

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d .
- $\text{score} = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$
- 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

Document frequency

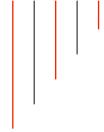
- **Rare terms are more informative than frequent terms**
 - Recall stop words
- Consider a term in the query that is rare in the collection
 - e.g., *arachnocentric*

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
 - $0 < \text{df}_t \leq N$

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 $\log (N/\text{df}_t)$ 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	

$$idf_t = \log_{10} (N/df_t)$$

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
<i>insurance</i>	10440	3997
<i>try</i>	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.

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

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

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

Relevance or Score for a document d given a query q

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

- 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...*

Informatics 225

Computer Science 221

Information Retrieval

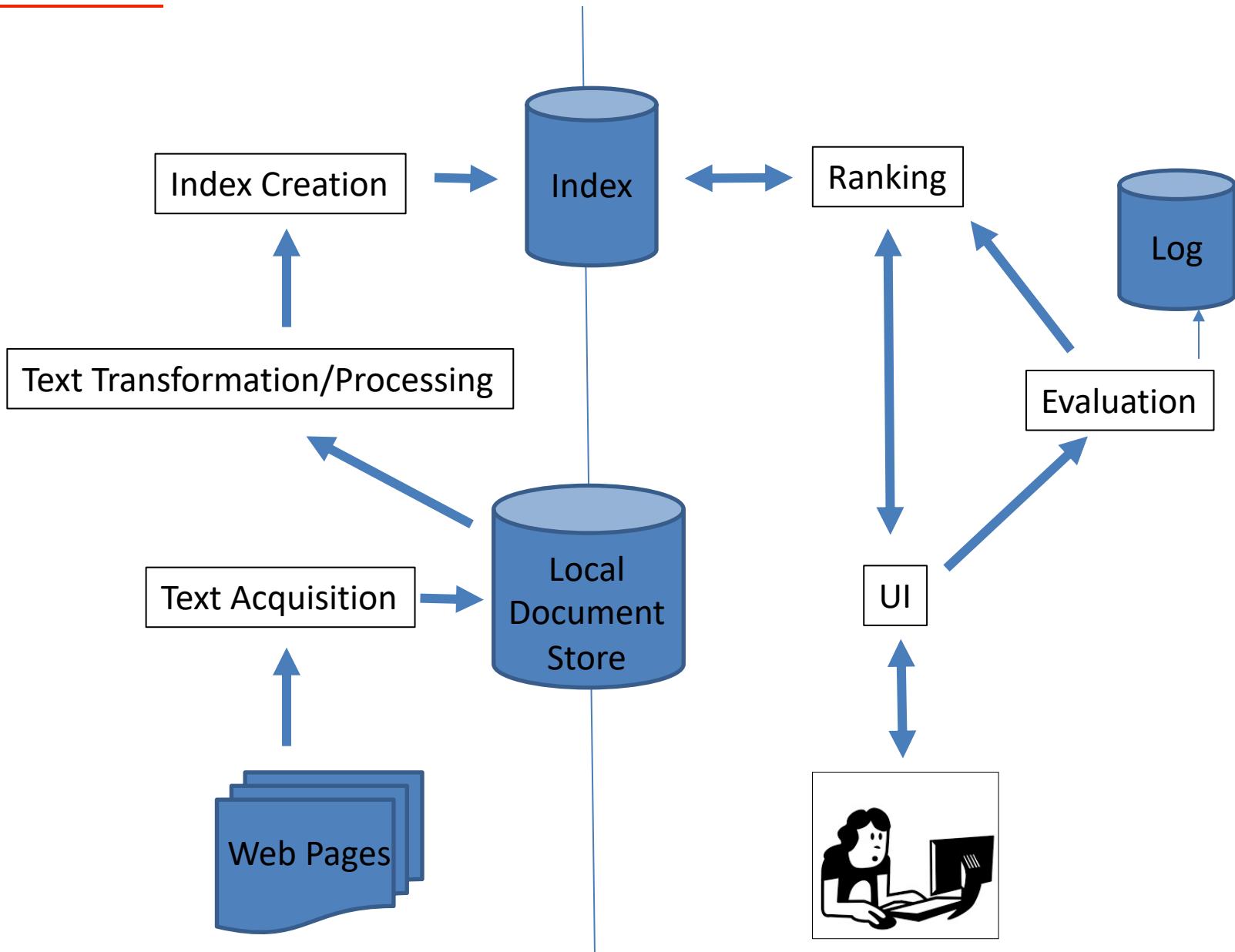
Lecture 21

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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

Preprocessing Steps



Vector space model

Binary → count → weight matrix



Term-document incidence → Term-document count matrix → **term-document weight matrix**

Binary → count → weight matrix

Term-document incidence → Term-document count matrix → term-document 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 $\in \mathbb{R}^{|V|}$



Documents as vectors

- So we have a $|V|$ -dimensional vector space



Documents as vectors

- So we have a $|V|$ -dimensional vector space
- Terms are axes of the space



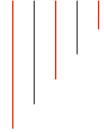
Documents as vectors

- So we have a $|V|$ -dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space



Documents as vectors

- So we have a $|V|$ -dimensional 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*



Documents as vectors

- So we have a $|V|$ -dimensional 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.



Queries as vectors

- **Key idea 1:** Do the same for queries: represent them as vectors in the space



Queries as vectors

- **Key idea 1:** Do the same for queries: represent them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space



Queries as vectors

- **Key idea 1:** Do the same for queries: represent them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity \approx inverse of distance

Queries as vectors



- **Key idea 1:** Do the same for queries: represent them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity \approx inverse of distance
- Recall:
 - We want to get away from the you're-either-in-or-out Boolean model.
 - Instead: rank more relevant documents higher than less relevant documents

Formalizing vector space proximity



- First thing that comes to mind: **distance between two points**
 - (= distance between the end points of the two vectors)

Formalizing vector space proximity

- First thing that comes to mind: distance between two points
 - (= distance between the end points of the two vectors)
- Euclidean distance?

Formalizing vector space proximity



- First thing that comes to mind: distance between two points
 - (= distance between the end points of the two vectors)
- Euclidean distance?
 - Euclidean distance is a bad idea . . .

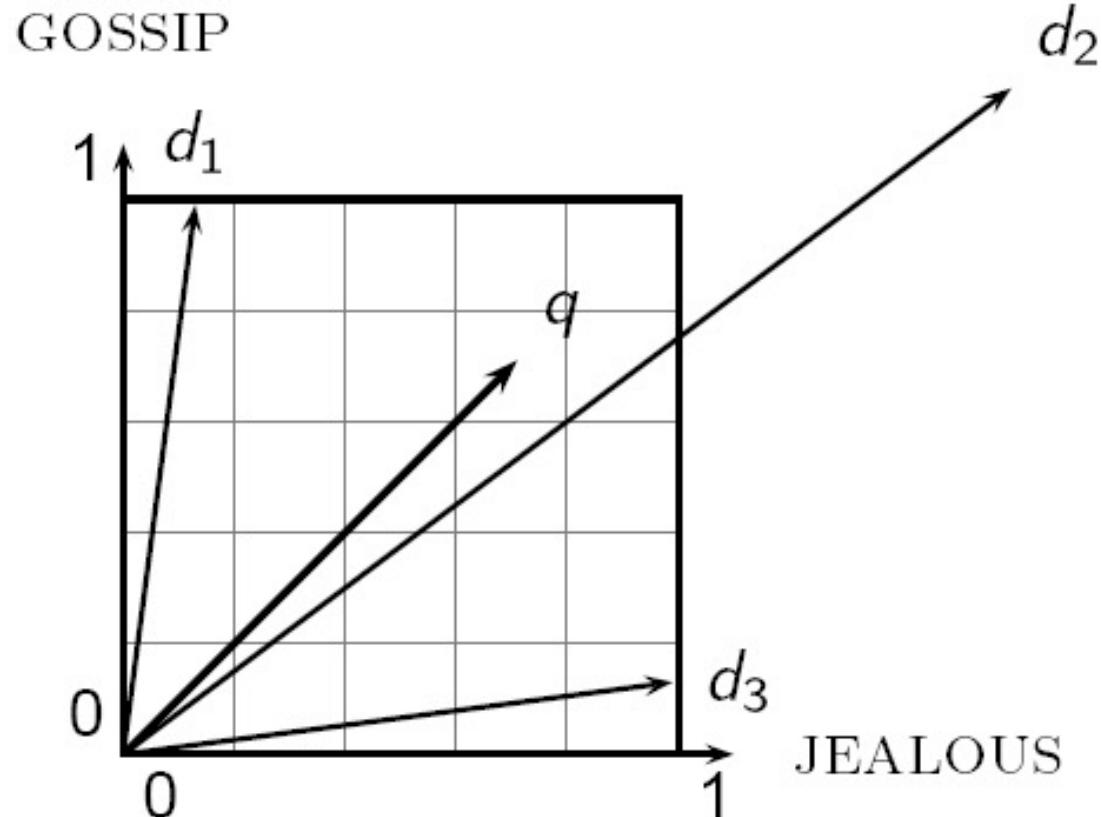
Formalizing vector space proximity



- First thing that comes to mind: distance between two points
 - (= distance between the end points of the two vectors)
- Euclidean distance?
 - Euclidean distance is a bad idea . . .
 . . . because Euclidean distance is large for vectors of different lengths.
 and queries and documents will (usually) have very different lengths...

Why Euclidean distance is (often) a bad idea

The Euclidean distance between \vec{q} and $\vec{d_2}$ is large even though the distribution of terms in the query \vec{q} and the distribution of terms in the document $\vec{d_2}$ are very similar.



What can be better?



- Thought experiment: take a document d and append it to itself. Call this document d' .

What can be better?



- Thought experiment: take a document d and append it to itself. Call this document d' .
 - “Semantically” d and d' have the same content

What can be better?



- Thought experiment: take a document d and append it to itself. Call this document d' .
 - “Semantically” d and d' have the same content
 - The Euclidean distance between the two documents can be quite large : *bad!*

Use angle instead of distance



- Thought experiment: take a document d and append it to itself. Call this document d' .
 - “Semantically” d and d' have the same content
 - The Euclidean distance between the two documents can be quite large : *bad!*
 - The angle between the two documents is 0, corresponding to maximal similarity : *good!*

Use angle instead of distance

- Thought experiment: take a document d and append it to itself. Call this document d' .
 - “Semantically” d and d' have the same content
 - The Euclidean distance between the two documents can be quite large : *bad!*
 - The angle between the two documents is 0, corresponding to maximal similarity : *good!*
- **Key idea:** Rank documents according to the angles between documents and query.

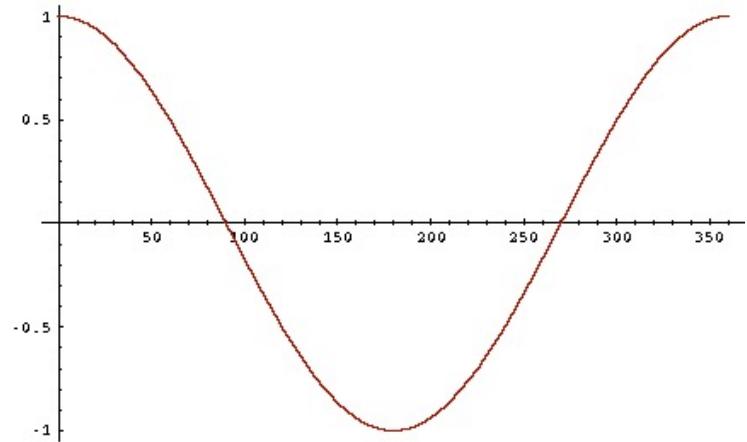
From angles to cosines



- The following two notions are equivalent.
 - Rank documents in decreasing order of angle(query, document)
 - Rank documents in increasing order of cosine(query, document)

From angles to cosines

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



cosine(query, document)

Dot product

Unit vectors

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

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

$\cos(\vec{q}, \vec{d})$ is the cosine similarity of \vec{q} and \vec{d} ... or,
equivalently, the cosine of the angle between \vec{q} and \vec{d} .



Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L₂ norm:

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

Length normalization



- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L₂ norm:

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

- Dividing a vector by its L₂ norm makes it a unit (length) vector
(on surface of unit hypersphere)

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L₂ norm:

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

- Dividing a vector by its L₂ norm makes it a unit (length) vector (*on surface of unit hypersphere*)
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
 - Long and short documents now have comparable weights



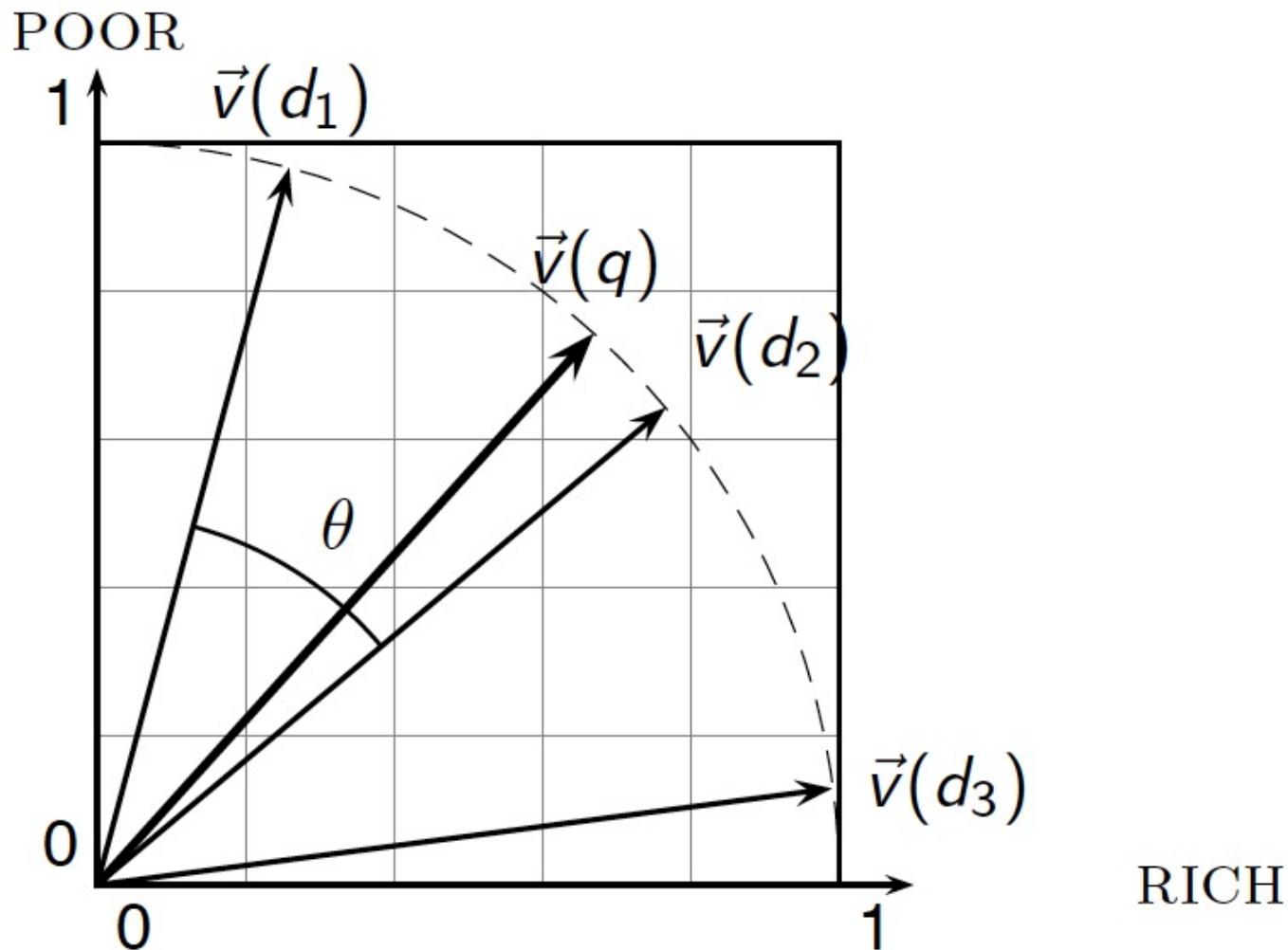
Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

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

for q , d length-normalized.

Cosine similarity illustrated



Computing cosine scores

COSINESCORE(q)

- 1 *float Scores[N] = 0*
- 2 *float Length[N]*
- 3 **for each** query term t
- 4 **do** calculate $w_{t,q}$ and fetch postings list for t
- 5 **for each** pair($d, tf_{t,d}$) in postings list
- 6 **do** $Scores[d] += w_{t,d} \times w_{t,q}$
- 7 Read the array $Length$
- 8 **for each** d
- 9 **do** $Scores[d] = Scores[d] / Length[d]$
- 10 **return** Top K components of $Scores[]$

Cosine similarity amongst 3 documents

- How similar are the novels?

- SaS: *Sense and Sensibility*
- PaP: *Pride and Prejudice*
- WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Cosine similarity amongst 3 documents

- How similar are the novels?
 - **SaS**: *Sense and Sensibility*
 - **PaP**: *Pride and Prejudice*
 - **WH**: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

3 documents example contd.

Log frequency weighting

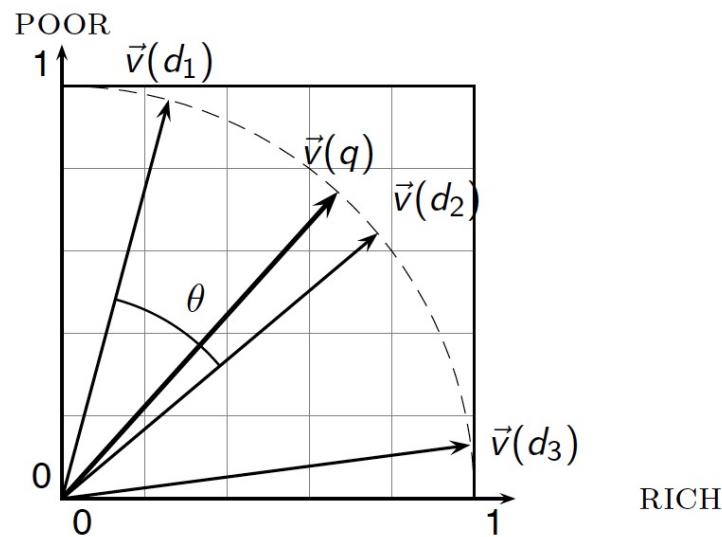
term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

*Documents
are already
in the surface of a
hypersphere, so*

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



3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

$$\cos(\text{SaS}, \text{PaP}) \approx$$

$$0.789 \times 0.832 + 0.515 \times 0.557 + 0.335 \times 0.0 + 0.0 \times 0.0$$

$$\approx 0.94$$

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

$$\cos(\text{SaS}, \text{PaP}) \approx$$

$$0.789 \times 0.832 + 0.515 \times 0.557 + 0.335 \times 0.0 + 0.0 \times 0.0$$

$$\approx 0.94$$

$$\cos(\text{SaS}, \text{WH}) \approx 0.79$$

$$\cos(\text{PaP}, \text{WH}) \approx 0.69$$

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.557	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

$$\cos(\text{SaS}, \text{PaP}) \approx$$

$$0.789 \times 0.832 + 0.515 \times 0.557 + 0.335 \times 0.0 + 0.0 \times 0.0$$

$$\approx 0.94$$

$$\cos(\text{SaS}, \text{WH}) \approx 0.79$$

$$\cos(\text{PaP}, \text{WH}) \approx 0.69$$

Why do we have $\cos(\text{SaS}, \text{PaP}) > \cos(\text{SaS}, \text{WH})$?

tf-idf weighting has many variants

Term frequency	Document frequency	Normalization
n (natural) $tf_{t,d}$	n (no) 1	n (none) 1
I (logarithm) $1 + \log(tf_{t,d})$	t (idf) $\log \frac{N}{df_t}$	c (cosine) $\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented) $0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf) $\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique) $1/u$
b (boolean) $\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		b (byte size) $1/CharLength^\alpha, \alpha < 1$
L (log ave) $\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$		

Itc : the general standard that is usually adopted

In parenthesis: acronyms for weight schemes.

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq*, using the acronyms from the previous table

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq*, using the acronyms from the previous table
- A very standard weighting scheme is: Inc.ltc

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation $ddd.ooo$, using the acronyms from the previous table
- **A very standard weighting scheme is: Inc.ltc**
- **Document:** logarithmic tf (l as first character), no idf and cosine normalization
- **Query:** logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization ...

tf-idf example: Inc.ltc (remember : ddd.ddd)

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query							Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'liz e	tf-raw	tf-wt	wt	n'liz e		
auto	0						1					
best	1						0					
car	1						1					
insurance	1						2					

tf-idf example: Inc.ltc (remember : ddd.ddd)

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query							Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'liz e	tf-raw	tf-wt	wt	n'liz e		
auto	0	0					1	1				
best	1	1					0	0				
car	1	1					1	1				
insurance	1	1					2	1.3				

tf-idf example: Inc.ltc (remember : ddd.ddd)

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query							Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'liz e	tf-raw	tf-wt	wt	n'liz e		
auto	0	0	5000	2.3			1	1				
best	1	1	50000	1.3			0	0				
car	1	1	10000	2.0			1	1				
insurance	1	1	1000	3.0			2	1.3				

tf-idf weighting has many variants

Term frequency	Document frequency	Normalization
n (natural) $tf_{t,d}$	n (no) 1	n (none) 1
I (logarithm) $1 + \log(tf_{t,d})$	t (idf) $\log \frac{N}{df_t}$	c (cosine) $\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented) $0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf) $\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique) $1/u$
b (boolean) $\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		b (byte size) $1/CharLength^\alpha, \alpha < 1$
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$	

lfc : standard

In parenthesis: acronyms for weight schemes.

tf-idf example: Inc.ltc (remember : ddd.ddd)

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query							Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'liz e	tf-raw	tf-wt	wt	n'liz e		
auto	0	0	5000	2.3	0		1	1	1			
best	1	1	50000	1.3	1.3		0	0	0			
car	1	1	10000	2.0	2.0		1	1	1			
insurance	1	1	1000	3.0	3.0		2	1.3	1.3			

tf-idf example: Inc.ltc

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query							Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'liz e	tf-raw	tf-wt	wt	n'liz e		
auto	0	0	5000	2.3	0	0	1	1	1	0.52		
best	1	1	50000	1.3	1.3	0.34	0	0	0	0		
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52		
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68		

$$\text{Doc length} = \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$$

tf-idf example: Inc.ltc

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query							Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'liz e	tf-raw	tf-wt	wt	n'liz e		
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0	
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0	
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27	
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53	

$$\text{Score} = 0+0+0.27+0.53 = 0.8$$

Summary – vector space ranking



1. Represent the query as a weighted tf-idf vector
2. Represent each document as a weighted tf-idf vector

Summary – vector space ranking

1. Represent the query as a weighted tf-idf vector
2. Represent each document as a weighted tf-idf vector
3. Compute the cosine similarity score for the query vector and each document vector

Summary – vector space ranking

1. Represent the query as a weighted tf-idf vector
2. Represent each document as a weighted tf-idf vector
3. Compute the cosine similarity score for the query vector and each document vector
4. Rank documents with respect to the query by score
5. Return the top K (e.g., $K = 10$) to the user

Informatics 225

Computer Science 221

Information Retrieval

Lecture 22

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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.

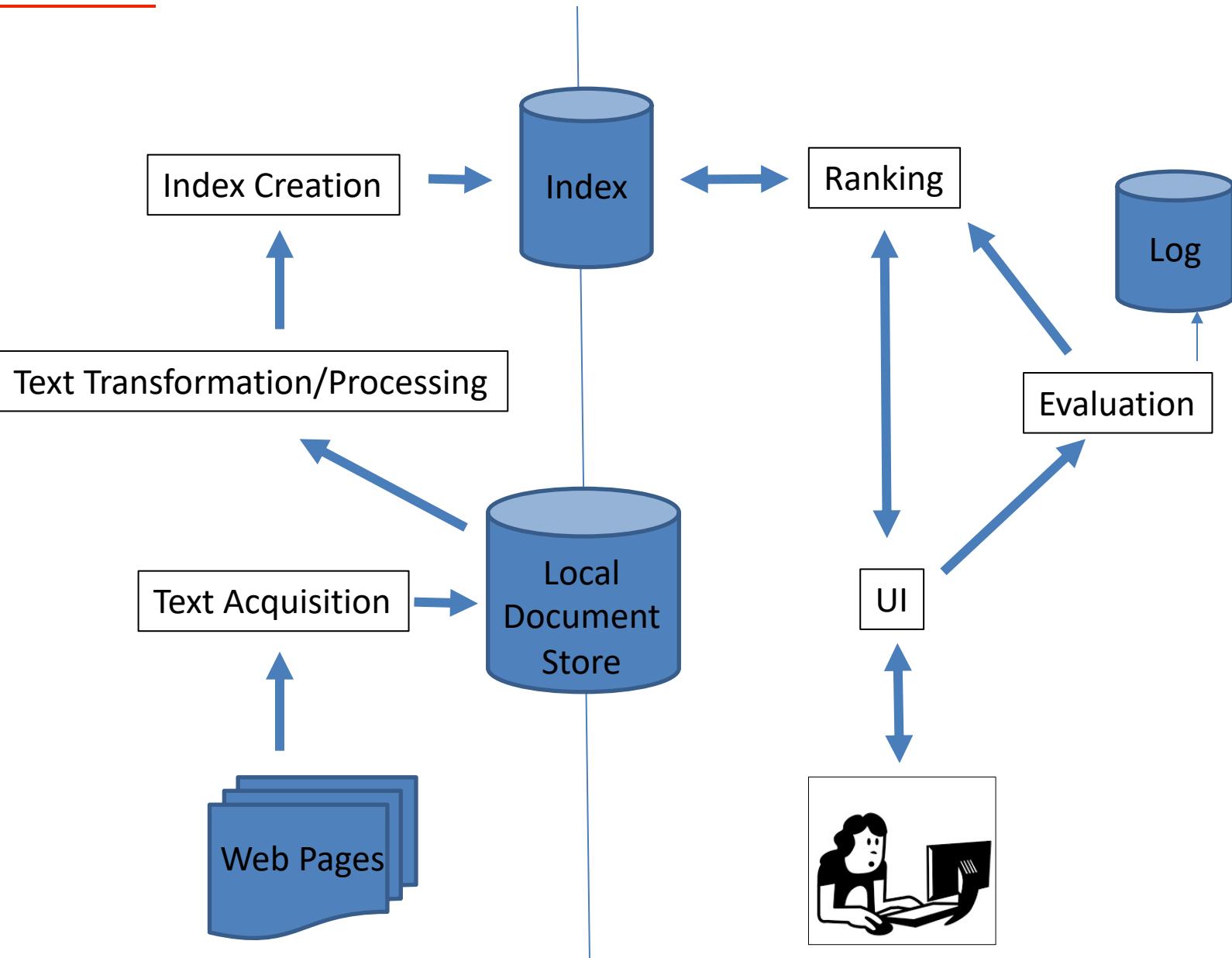
Scoring and result assembly

Information Retrieval

These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, and additional materials from Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel and Sascha Rothe

Architecture

Preprocessing Steps



Recap: tf-idf weighting

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

$$w_{t,d} = (1 + \log_{10} tf_{t,d}) \times \log_{10}(N / df_t)$$

- Best known weighting scheme in information retrieval
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Recap: Queries as vectors



- **Key idea 1:** Do the same for queries: represent them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity \approx inverse of distance

Recap: cosine(query,document)

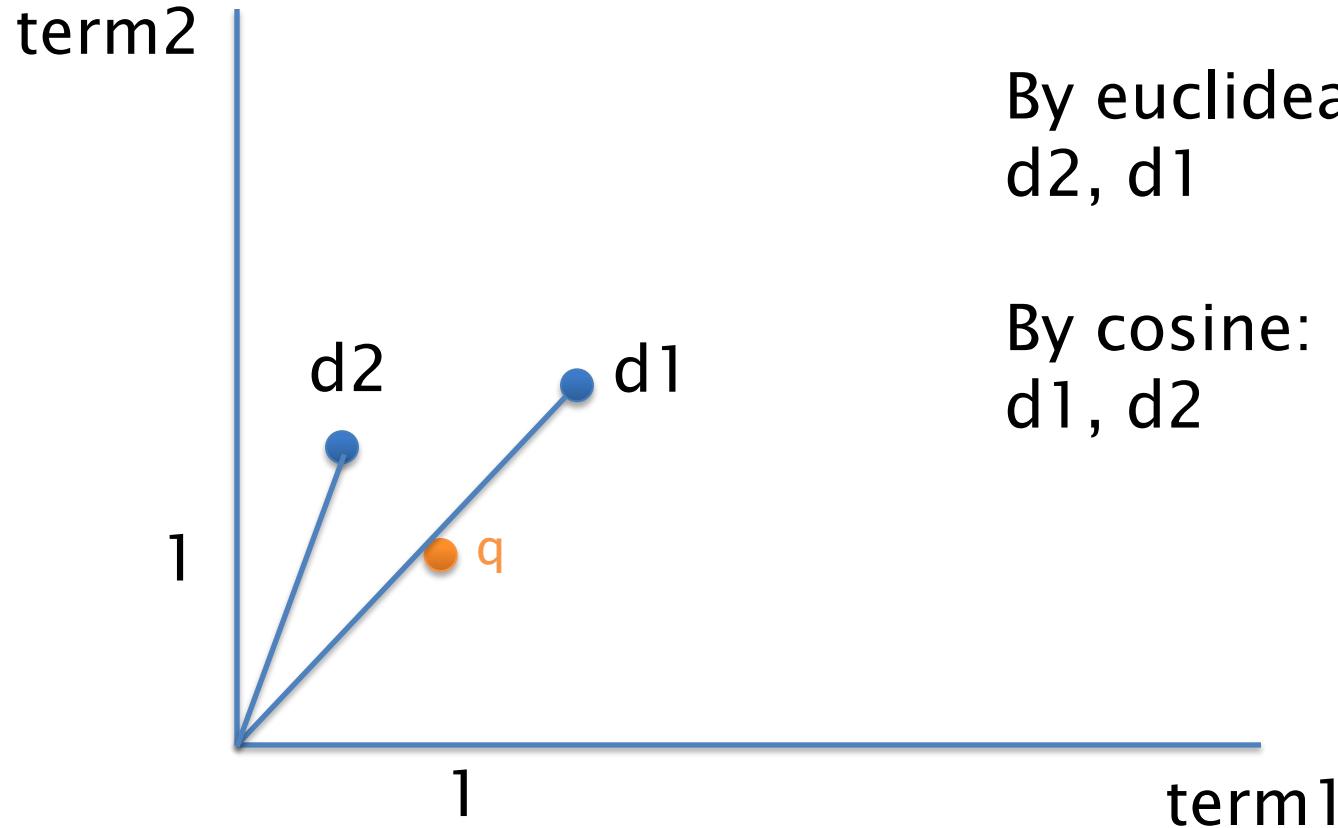
Dot product

Unit vectors

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

$\cos(\vec{q}, \vec{d})$ is the cosine similarity of \vec{q} and \vec{d} ... or,
equivalently, the cosine of the angle between \vec{q} and \vec{d} .

Simple tf-idf vs. cosine scoring



By euclidean tf-idf:
 d_2, d_1

By cosine:
 d_1, d_2

Now...

- Speeding up vector space ranking
- Starting to put together a complete search system
 - Will require us to consider miscellaneous topics and **heuristics**

Computing cosine scores

COSINESCORE(q)

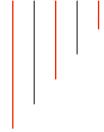
- 1 *float Scores[N] = 0*
- 2 *float Length[N]*
- 3 **for each** query term t
- 4 **do** calculate $w_{t,q}$ and fetch postings list for t
- 5 **for each** pair($d, tf_{t,d}$) in postings list
- 6 **do** $Scores[d] += w_{t,d} \times w_{t,q}$
- 7 Read the array $Length$
- 8 **for each** d
- 9 **do** $Scores[d] = Scores[d]/Length[d]$
- 10 **return** Top K components of $Scores[]$

Efficient cosine ranking



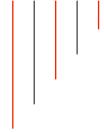
- Find the K docs in the collection “nearest” to the query $\Rightarrow K$ largest query-doc cosines.

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.

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.

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?**

Efficient cosine ranking



- What we're doing in effect: solving the K -nearest neighbor problem for a query vector

Efficient cosine ranking



- What we're doing in effect: solving the K -nearest neighbor problem for a query vector
- In general, we do not know how to do this efficiently for high-dimensional spaces

Efficient cosine ranking



- What we're doing in effect: solving the K -nearest neighbor problem for a query vector
- In general, we do not know how to do this efficiently for high-dimensional spaces
- But we don't need to solve for the very high dimensional spaces in the context of websearch! Query vectors are highly sparse!

Efficient cosine ranking

- What we're doing in effect: solving the K -nearest neighbor problem for a query vector
- In general, we do not know how to do this efficiently for high-dimensional spaces
- But we don't need to solve for the very high dimensional spaces in the context of websearch! Query vectors are highly sparse!
 - The problem is solvable for short queries, and standard indexes support this well

Special case – unweighted queries

- No weighting on query terms
 - Assume each query term occurs only once
- Then for ranking, don't need to normalize query vector

Computing the K largest cosines: selection vs. sorting

- Typically we want to retrieve the top K docs (in the cosine ranking for the query)
 - And not to totally order all docs in the collection

Computing the K largest cosines: selection vs. sorting

- Typically we want to retrieve the top K docs (in the cosine ranking for the query)
 - And not to totally order all docs in the collection
- Can we pick off docs with K highest cosines?

Computing the K largest cosines: selection vs. sorting

- Typically we want to retrieve the top K docs (in the cosine ranking for the query)
 - And not to totally order all docs in the collection
- Can we pick off docs with K highest cosines?
- Let $J = \text{number of docs with nonzero cosines}$
 - We seek the K best of these J

Use heap for selecting top K



- Binary tree in which each node's value > the values of children
- Takes $2J$ operations to construct, then each of K “winners” read off in $2\log J$ steps.
- For $J=1M$, $K=100$, this is about $\sim 10\%$ of the cost of sorting!

Bottlenecks

- Primary computational bottleneck in scoring: cosine computation

Bottlenecks

- Primary computational bottleneck in scoring: cosine computation
- Can we avoid all this computation?

Bottlenecks

- Primary computational bottleneck in scoring: cosine computation
- Can we avoid all this computation?
- Yes, but may sometimes get it wrong

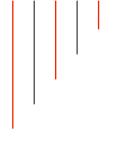
Bottlenecks

- Primary computational bottleneck in scoring: cosine computation
- Can we avoid all this computation?
- Yes, but may sometimes get it wrong
 - a doc *not* in the top K *may* creep into the list of K output docs

Bottlenecks

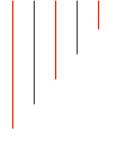
- Primary computational bottleneck in scoring: cosine computation
- Can we avoid all this computation?
- Yes, but may sometimes get it wrong
 - a doc *not* in the top K **may** creep into the list of K output docs
 - **Is this such a bad thing?**

Cosine similarity...



- User has a task and a query formulation

Cosine similarity...



- User has a task and a query formulation
- Cosine matches docs to query

Cosine similarity is only a proxy!



- User has a task and a query formulation
- Cosine matches docs to query
- Thus cosine is anyway a proxy for user happiness

Cosine similarity is only a proxy!



- User has a task and a query formulation
- Cosine matches docs to query
- Thus cosine is anyway a proxy for user happiness
- **If we get a list of K docs “close” to the top K by cosine measure, should be ok**

Generic approach



- Find a set A of *contenders*, with $K < |A| \ll N$



Generic approach

- Find a set A of *contenders*, with $K < |A| \ll N$
 - A does not necessarily contain the top K ,
 - **but has many docs from among the top K**
 - Return the top K docs in A

Generic approach



- Find a set A of *contenders*, with $K < |A| \ll N$
 - A does not necessarily contain the top K ,
 - but has many docs from among the top K
 - Return the top K docs in A
- Think of A as pruning non-contenders

Generic approach



- Find a set A of *contenders*, with $K < |A| \ll N$
 - A does not necessarily contain the top K ,
 - but has many docs from among the top K
 - Return the top K docs in A
- Think of A as pruning non-contenders
- The same approach is also used for other (non-cosine) scoring functions
- Will look at a few schemes following this approach

Index elimination

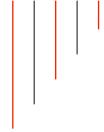


- Basic algorithm for cosine computation only considers docs containing at least one query term

Index elimination

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

Index elimination: High-idf query terms only



- For a query such as *catcher in the rye*
- Only accumulate scores from *catcher* and *rye*

Index elimination: High-idf query terms only

- For a query such as *catcher in the rye*
- Only accumulate scores from *catcher* and *rye*
- Intuition: *in* and *the* contribute little to the scores and so don't alter rank-ordering much

Index elimination: High-idf query terms only

- For a query such as *catcher in the rye*
- Only accumulate scores from *catcher* and *rye*
- Intuition: *in* and *the* contribute little to the scores and so don't alter rank-ordering much
- **Benefit:**
 - Postings of low-idf terms have many docs → these (many!!!) docs get eliminated from set A of contenders

Index elimination: Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list

Index elimination: Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms (perhaps all terms!)

Index elimination: Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms (perhaps all terms!)
 - Say, at least 3 out of 4

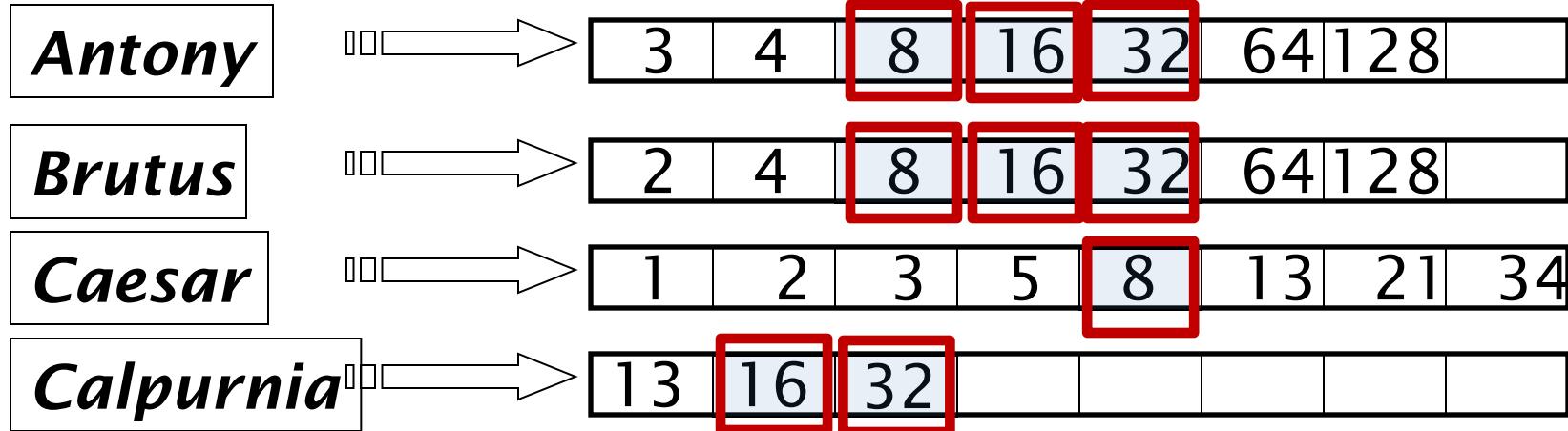
Index elimination: Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms (perhaps all terms!)
 - Say, at least 3 out of 4
 - Imposes a “soft conjunction” on queries seen on web search engines (early Google)

Index elimination: Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms (perhaps all terms!)
 - Say, at least 3 out of 4
 - Imposes a “soft conjunction” on queries seen on web search engines (early Google)
 - May end up with fewer than K candidates if too restrictive in the number of query terms. Add another heuristic on top: *you can iterate!*

Example 3 of 4 query terms



Scores only computed for docs 8, 16 and 32.

If you only want to show 2 docs to the user, it is enough.
If you need more, you iterate and now compute scores
for 2 query terms also.

Index elimination: Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms (perhaps all terms!)
 - Say, at least 3 out of 4
 - Imposes a “soft conjunction” on queries seen on web search engines (early Google)
 - May end up with fewer than K candidates if too restrictive in the number of query terms. Add another heuristic on top: *you can iterate!*
- Easy to implement in postings traversal

Informatics 225

Computer Science 221

Information Retrieval

Lecture 23

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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

Preprocessing Steps

Index Creation



Index



Ranking

Text Transformation/Processing



Text Acquisition

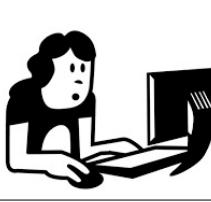


Local Document Store

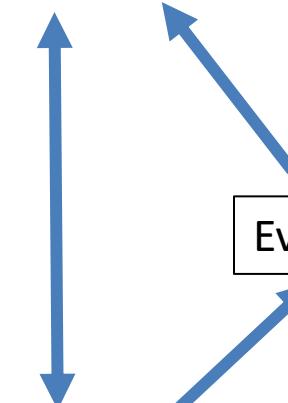
Web Pages



Querying Process



UI



Evaluation

Log



Generic approach



- Find a set A of *contenders*, with $K < |A| \ll N$
 - A does not necessarily contain the top K ,
 - but has many docs from among the top K
 - Return the top K docs in A
- Think of A as pruning non-contenders
- The same approach is also used for other (non-cosine) scoring functions
- Will look at a few schemes following this approach

Index elimination

- Basic algorithm for cosine computation only considers docs containing at least one query term
- Take this heuristic 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

Champion lists



- Precompute for each dictionary term t , the r docs of highest weight in t 's postings
 - Call this the champion list for t
 - (aka fancy list or top docs for t)

Champion lists



- Precompute for each dictionary term t , the r docs of highest weight in t 's postings
 - Call this the champion list for t
 - (aka fancy list or top docs for t)
- Note that r has to be chosen at index build time
 - Thus, it's possible that $r < K$

Champion lists



- Precompute for each dictionary term t , the r docs of highest weight in t 's postings
 - Call this the champion list for t
 - (aka fancy list or top docs for t)
- Note that r has to be chosen at index build time
 - Thus, it's possible that $r < K$
- At query time, only compute the complete scores for docs in the champion list of some query term

Champion lists



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

Static quality scores



- We want top-ranking documents to be both *relevant* and *authoritative*

Static quality scores



- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores

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

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 ←
 - Many bitly's, diggs or del.icio.us marks ←
 - (Pagerank, HITS) ←

Quantitative



Modeling authority

- Assign to each document **a *query-independent quality score*** in $[0,1]$ to each document d
 - Denote this by $g(d)$

Modeling authority

- Assign to each document a *query-independent quality score* in $[0,1]$ to each document d
 - Denote this by $g(d)$
- Thus, a quantity like the number of citations is scaled into $[0,1]$
 - Informal exercise: suggest a formula for this.

Net relevance score



- Consider a simple total score combining cosine relevance and authority

Net relevance score

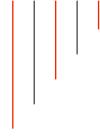
- Consider a simple total score combining cosine relevance and authority
- $\text{net-score}(q,d) = g(d) + \cosine(q,d)$
 - Can use some other linear combination: $g(d) + \alpha * \cosine(q,d)$
 - Indeed, any function of the two “signals” of user happiness

Net relevance score



- Consider a simple total score combining cosine relevance and authority
- $\text{net-score}(q,d) = g(d) + \cosine(q,d)$
 - Can use some other linear combination: $g(d) + \alpha * \cosine(q,d)$
 - Indeed, any function of the two “signals” of user happiness
- Now we seek the top K docs by net score

Top K by net score – fast methods



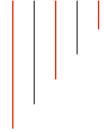
- First idea: Order all postings by $g(d)$

Top K by net score – fast methods



- First idea: Order all postings by $g(d)$
- Key: this is a common ordering for all postings

Top K by net score – fast methods



- First idea: Order all postings by $g(d)$
- Key: this is a common ordering for all postings
- Thus, can concurrently traverse query terms' postings for
 - Postings intersection
 - Cosine score computation



Why order postings by $g(d)$?

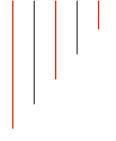
- Under $g(d)$ -ordering, top-scoring docs likely to appear early in postings traversal (remember : $g(d) + \alpha * \text{cosine}(q,d)$)

Why order postings by $g(d)$?



- Under $g(d)$ -ordering, top-scoring docs likely to appear early in postings traversal
- In time-bound applications (say, we have to return whatever search results we can in ~50 ms), this allows us to stop postings traversal early
 - Short of computing complete scores for all docs in postings

Champion lists in $g(d)$ -ordering



- Can combine champion lists with $g(d)$ -ordering



Champion lists in $g(d)$ -ordering

- Can combine champion lists with $g(d)$ -ordering
- Maintain for each term a champion list of the r docs with highest $g(d) + \text{tf-idf}_{td}$

Champion lists in $g(d)$ -ordering



- Can combine champion lists with $g(d)$ -ordering
- Maintain for each term a champion list of the r docs with highest $g(d) + \text{tf-idf}_{td}$
- Seek top- K results from only the docs in these champion lists



Another heuristic: High and low lists

- For each term, we **maintain two disjoint sets: *high* and *low lists***
 - Think of *high* as the champion list



Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
 - Think of *high* as the champion list
- When traversing postings on a query, traverse *high* lists first



Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
 - Think of *high* as the champion list
- When traversing postings on a query, traverse *high* lists first
 - If we get more than K docs, select the top K and stop
 - Else proceed to get docs from the *low* lists

Another heuristic: High and low lists

- For each term, we maintain two disjoint sets: *high* and *low lists*
 - Think of *high* as the champion list
- When traversing postings on a query, traverse *high* lists first
 - If we get more than K docs, select the top K and stop
 - Else proceed to get docs from the *low* lists
- Can be used even for simple cosine scores, without global quality $g(d)$

Another heuristic: High and low lists

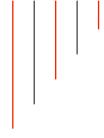
- For each term, we maintain two disjoint sets: *high* and *low lists*
 - Think of *high* as the champion list
- When traversing postings on a query, traverse *high* lists first
 - If we get more than K docs, select the top K and stop
 - Else proceed to get docs from the *low* lists
- Can be used even for simple cosine scores, without global quality $g(d)$
- A means for segmenting index into two tiers



Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough

Impact-ordered postings



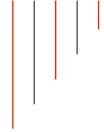
- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$

Impact-ordered postings



- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!

Impact-ordered postings



- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!
- How do we compute scores in order to pick off top K ?
 - Two ideas follow



Idea 1. Early termination

- When traversing t 's postings, stop early after either
 - a fixed number of r docs
 - $wf_{t,d}$ drops below some threshold

Idea 1. Early termination



- When traversing t 's postings, stop early after either
 - a fixed number of r docs
 - $wf_{t,d}$ drops below some threshold
- Take the union of the resulting sets of docs
 - One from the postings of each query term

Idea 1. Early termination



- When traversing t 's postings, stop early after either
 - a fixed number of r docs
 - $wf_{t,d}$ drops below some threshold
- Take the union of the resulting sets of docs
 - One from the postings of each query term
- Compute only the scores for docs in this union



Idea 2. idf-ordered terms

- When considering the postings of query terms

Idea 2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
 - High idf terms likely to contribute most to score

Idea 2. idf-ordered terms



- When considering the postings of query terms
- Look at them in order of decreasing idf
 - High idf terms likely to contribute most to score
- As we update score contribution from each query term
 - Stop if doc scores relatively unchanged

Idea 2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
 - High idf terms likely to contribute most to score
- As we update score contribution from each query term
 - Stop if doc scores relatively unchanged
- Can apply to cosine or some other net scores

Parametric and zone indexes



- Thus far, a doc has been a sequence of terms

Parametric and zone indexes



- Thus far, a doc has been a sequence of terms
- In fact documents have multiple parts, some with special semantics:
 - Author
 - Title
 - Date of publication
 - Language
 - Format
 - etc.

Parametric and zone indexes



- Thus far, a doc has been a sequence of terms
- In fact documents have multiple parts, some with special semantics:
 - Author
 - Title
 - Date of publication
 - Language
 - Format
 - etc.
- These constitute the metadata about a document

Fields



- We sometimes wish to search by these metadata
 - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*

Fields

- We sometimes wish to search by these metadata
 - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*
- Year = 1601 is an example of a field
- Also, author last name = shakespeare, etc.

Fields

- We sometimes wish to search by these metadata
 - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*
- Year = 1601 is an example of a field
- Also, author last name = shakespeare, etc.
- Field or parametric index: postings for each field value
 - Sometimes build range trees (e.g., for dates)

Fields

- We sometimes wish to search by these metadata
 - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*
- Year = 1601 is an example of a field
- Also, author last name = shakespeare, etc.
- Field or parametric index: postings for each field value
 - Sometimes build range trees (e.g., for dates)
- Field query typically treated as conjunction
 - (doc *must* be authored by shakespeare)

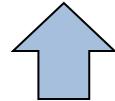
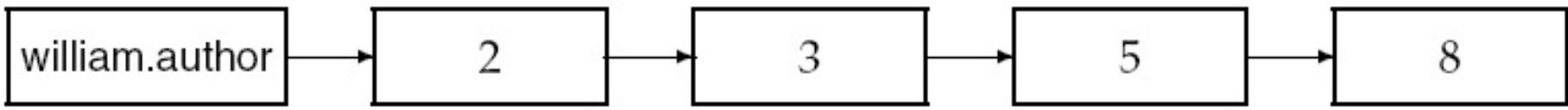
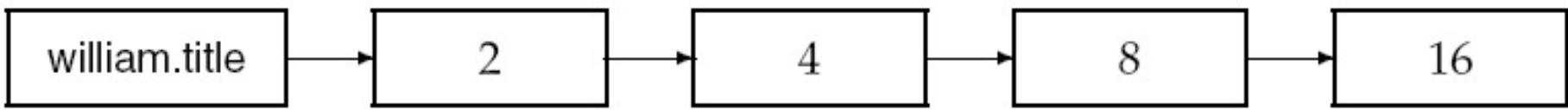
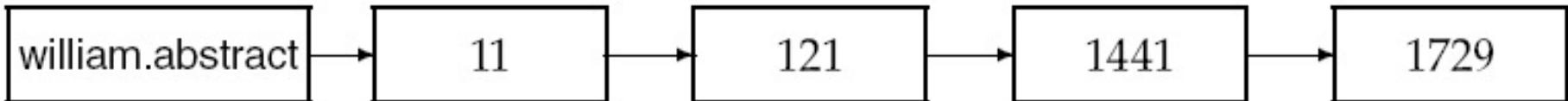


- A zone is a region of the doc that can contain an arbitrary amount of text, e.g.,
 - Title
 - Abstract
 - References ...

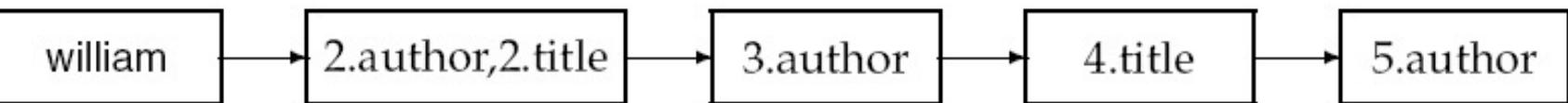
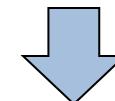


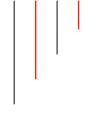
- A zone is a region of the doc that can contain an arbitrary amount of text, e.g.,
 - Title
 - Abstract
 - References ...
- Build inverted indexes on zones as well to permit querying
- E.g., “find docs with *merchant* in the title zone and matching the query *gentle rain*”

Example zone indexes



Encode zones in dictionary vs. postings.





Tiered indexes

- Break postings up into a hierarchy of lists
 - Most important
 - ...
 - Least important



Tiered indexes

- Break postings up into a hierarchy of lists
 - Most important
 - ...
 - Least important
- Can be done by $g(d)$ or another measure (zone weight, tf-idf, etc...)



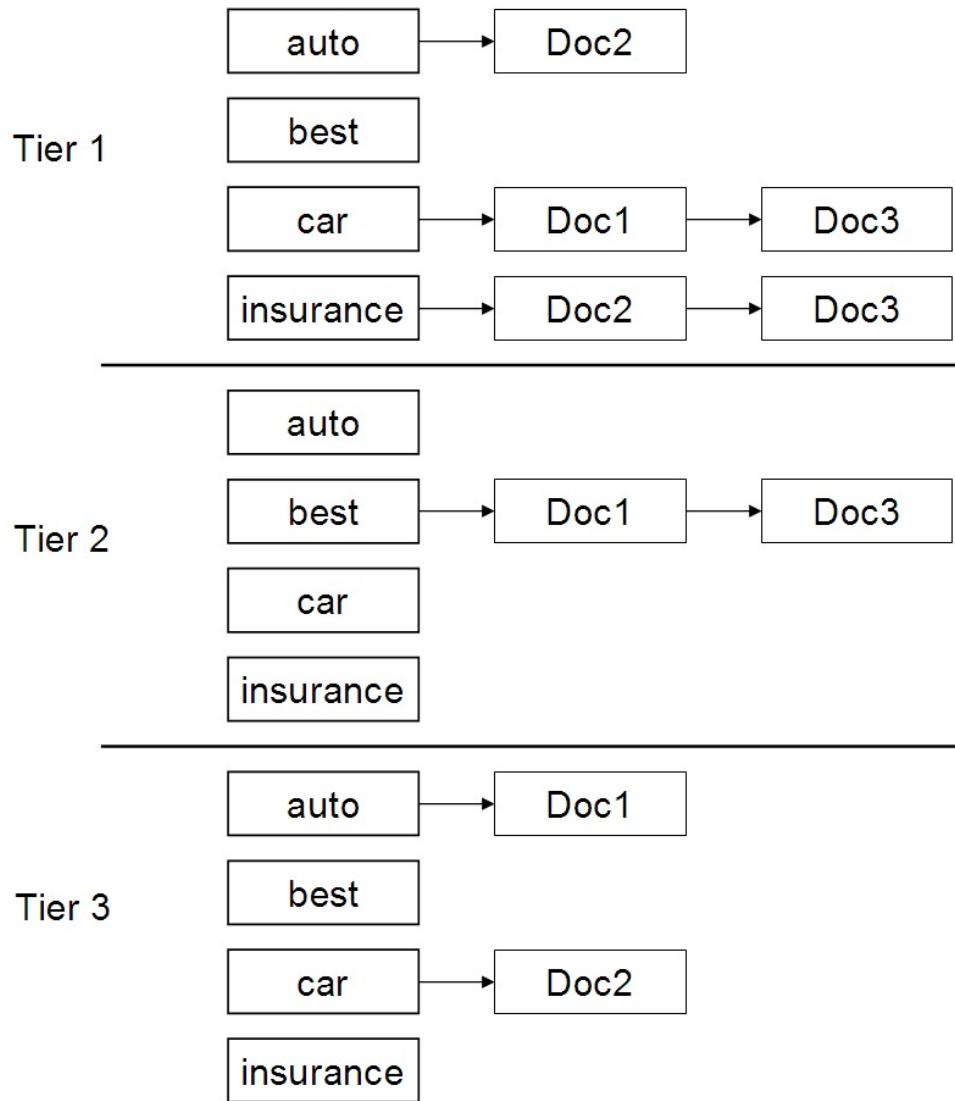
Tiered indexes

- Break postings up into a hierarchy of lists
 - Most important
 - ...
 - Least important
- Can be done by $g(d)$ or another measure (zone weight, tf-idf, etc...)
- Inverted index thus broken up into tiers of decreasing importance

Tiered indexes

- Break postings up into a hierarchy of lists
 - Most important
 - ...
 - Least important
- Can be done by $g(d)$ or another measure (zone weight, tf-idf, etc...)
- Inverted index thus broken up into tiers of decreasing importance
- At query time use top tier unless it fails to yield K docs
 - If so drop to lower tiers

Example tiered index



Query term proximity

- Free text queries: just a set of terms typed into the query box
 - common on the web

Query term proximity

- Free text queries: just a set of terms typed into the query box
 - common on the web
- Users prefer docs in which **query terms occur within close proximity of each other**

Query term proximity

- Free text queries: just a set of terms typed into the query box
 - common on the web
- Users prefer docs in which query terms occur within close proximity of each other
- **Let w be the smallest window** in a doc containing all query terms, e.g.,

Query term proximity

- Free text queries: just a set of terms typed into the query box
 - common on the web
- Users prefer docs in which query terms occur within close proximity of each other
- Let w be the smallest window in a doc containing all query terms, e.g.,
 - For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)

Query term proximity

- Free text queries: just a set of terms typed into the query box
 - common on the web
- Users prefer docs in which query terms occur within close proximity of each other
- Let w be the smallest window in a doc containing all query terms, e.g.,
 - For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)
 - Intuition : the smaller w is, the better the document matches the query.

Query term proximity

- Free text queries: just a set of terms typed into the query box
 - common on the web
- Users prefer docs in which query terms occur within close proximity of each other
- Let w be the smallest window in a doc containing all query terms, e.g.,
 - For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)
 - Intuition : the smaller w is, the better the document matches the query.
 - **Would like scoring function to take this into account : add weights!**

Architecture

Preprocessing Steps

Index Creation

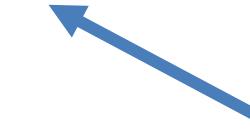


Index



Ranking

Text Transformation/Processing



Text Acquisition



Local Document Store

Web Pages

Querying Process

UI



Evaluation



Log

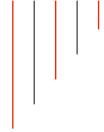


Query parsers



- Free text query from user may in fact spawn one or more queries to the indexes, e.g., query *rising interest rates*

Query parsers



- Free text query from user may in fact spawn one or more queries to the indexes, e.g., query *rising interest rates*
 - Run the query as a phrase query
 - If $<K$ docs contain the phrase *rising interest rates*, run the two phrase queries *rising interest* and *interest rates*
 - If we still have $<K$ docs, run the vector space query *rising interest rates*
 - Rank matching docs by vector space scoring

Query parsers



- Free text query from user may in fact spawn one or more queries to the indexes, e.g., query *rising interest rates*
 - Run the query as a phrase query
 - If $<K$ docs contain the phrase *rising interest rates*, run the two phrase queries *rising interest* and *interest rates*
 - If we still have $<K$ docs, run the vector space query *rising interest rates*
 - Rank matching docs by vector space scoring
- This sequence is issued by a **query parser**
 - Some method that will transform a user query into a stream of multiple queries.



Aggregate scores

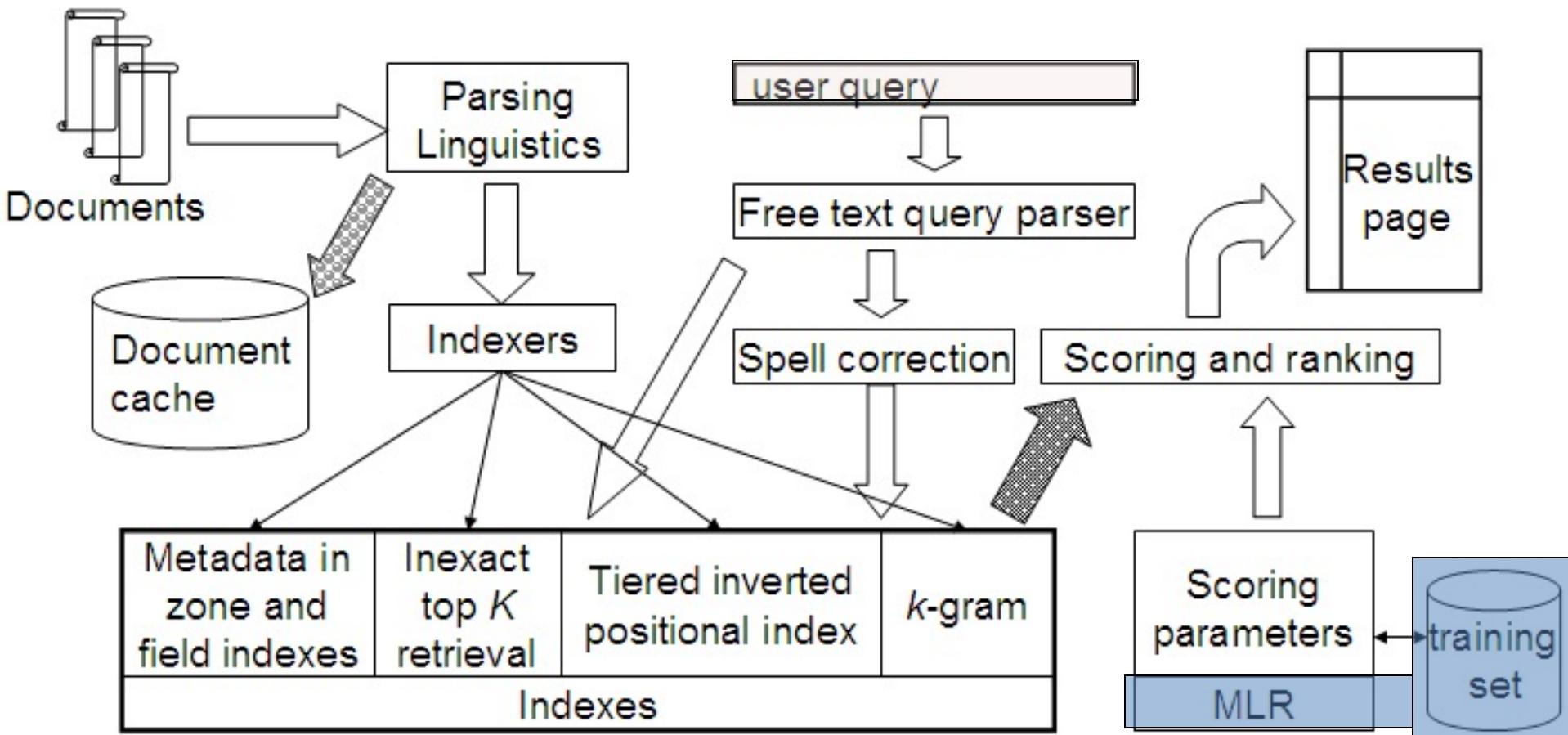
- We've seen that score functions can combine cosine, static quality, proximity, etc.

Aggregate scores



- We've seen that score functions can combine cosine, static quality, proximity, etc.
- How do we know the best combination?
 - *User surveys : ask real people to use your search engine (think MS3!)*
 - Some applications – expert-tuned
 - Hand-tune real-life systems is hard (e.g. tens or hundreds of possible threshold choices, what to consider or not, etc.)
 - Increasingly common technique: adaptive-learning based optimization

Putting it all together



Architecture

Preprocessing Steps

Index Creation



Index



Ranking

Text Transformation/Processing

Text Acquisition



Local Document Store

Web Pages

Querying Process



UI

Evaluation



Log



Informatics 225

Computer Science 221

Information Retrieval

Lecture 24

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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

Preprocessing Steps

Index Creation



Index



Ranking

Text Transformation/Processing

Text Acquisition



Local Document Store

Web Pages

Querying Process



UI

Evaluation



Log



Link Analysis

Information Retrieval

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



HUBS AND AUTHORITIES

Hyperlink-Induced Topic Search (HITS)



- In response to a query, instead of an ordered list of pages each meeting the query, **find two sets of inter-related pages:**
 - *Hub pages* are good lists of links on a subject.
 - e.g., “Bob’s list of cancer-related links.”
 - *Authority pages* occur recurrently on good hubs for the subject.

Hyperlink-Induced Topic Search (HITS)



- In response to a query, instead of an ordered list of pages each meeting the query, **find two sets of inter-related pages:**
 - *Hub pages* are good lists of links on a subject.
 - e.g., “Bob’s list of cancer-related links.”
 - *Authority pages* occur recurrently on good hubs for the subject.

Hyperlink-Induced Topic Search (HITS)



- In response to a query, instead of an ordered list of pages each meeting the query, **find two sets of inter-related pages:**
 - *Hub pages* are good lists of links on a subject.
 - e.g., “Bob’s list of cancer-related links.”
 - *Authority pages* occur recurrently on good hubs for the subject.
- Best suited for “broad topic” queries rather than for page-finding queries.
- Gets at a broader slice of common *opinion*.

Hubs and Authorities



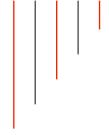
- A good hub page for a topic *points* to many authoritative pages for that topic.

Hubs and Authorities



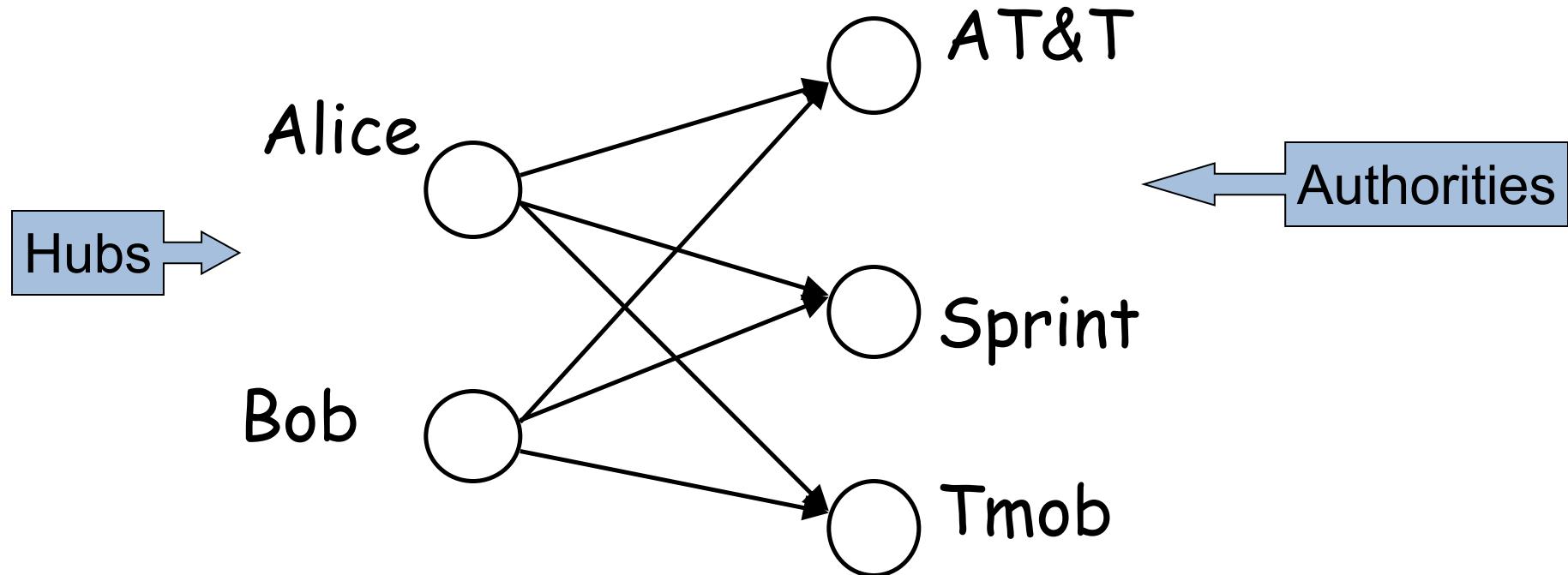
- A good hub page for a topic *points* to many authoritative pages for that topic.
- A good authority page for a topic is *pointed to* by many good hubs for that topic.

Hubs and Authorities



- A good hub page for a topic *points* to many authoritative pages for that topic.
- A good authority page for a topic is *pointed to* by many good hubs for that topic.
- **Circular definition** - will turn this into an iterative computation.

The hope



Mobile telecom companies

High-level scheme

- Extract from the web a base set of pages that *could* be good hubs or authorities.
- From these, identify a small set of top hub and authority pages;
→iterative algorithm.

Creating the Base set



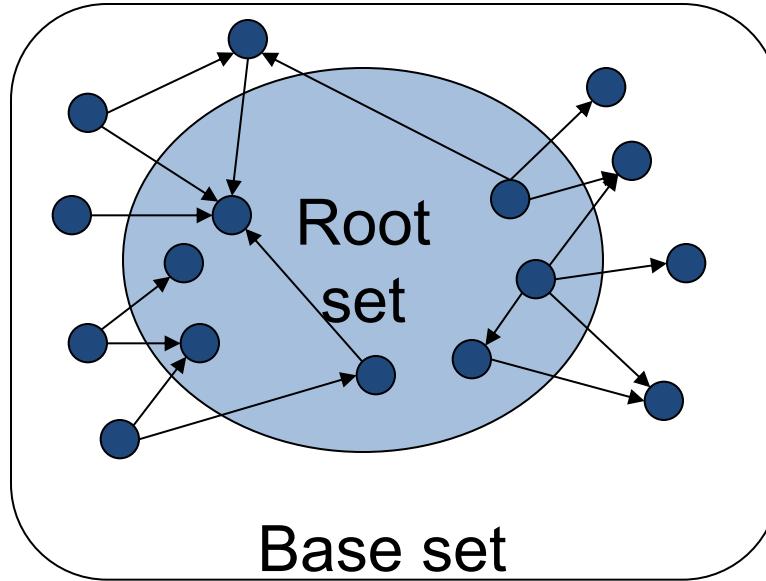
- Given text query (say *browser*), use a text index to get all pages containing *browser*.
 - Call this the root set of pages.

Creating the Base set



- Given text query (say *browser*), use a text index to get all pages containing *browser*.
 - Call this the root set of pages.
- Add in any page that either
 - points to a page in the root set, or
 - is pointed to by a page in the root set.
- Call this the base set.

Visualization



Get in-links (and out-links) from a *connectivity server*

Connectivity Server



- Support for fast queries on the web graph
 - Which URLs point to a given URL?
 - Which URLs does a given URL point to?

Connectivity Server



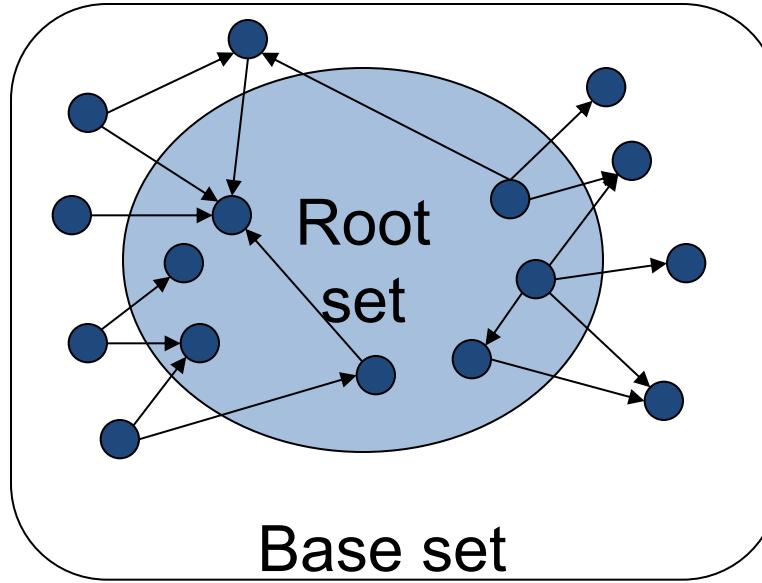
- Support for fast queries on the web graph
 - Which URLs point to a given URL?
 - Which URLs does a given URL point to?
- Stores mappings in memory from
 - URL to outlinks, URL to inlinks
 - Use sparse properties of the graph (do not build the entire matrix!)

Connectivity Server



- Support for fast queries on the web graph
 - Which URLs point to a given URL?
 - Which URLs does a given URL point to?
- Stores mappings in memory from
 - URL to outlinks, URL to inlinks
 - Use sparse properties of the graph (do not build the entire matrix!)
- Applications
 - Crawl control
 - Web graph analysis
 - *Connectivity, crawl optimization*
 - Link analysis (HITS, PageRank, ...)

Visualization



Get in-links (and out-links) from a *connectivity server*

Distilling hubs and authorities



- Compute, for each page x in the base set:
 - a **hub score** $h(x)$
 - and an **authority score** $a(x)$.

Distilling hubs and authorities



- Compute, for each page x in the base set:
 - a **hub score** $h(x)$
 - and an **authority score** $a(x)$.
- Initialize: for all x , $h(x) \leftarrow 1$; $a(x) \leftarrow 1$;

Distilling hubs and authorities



- Compute, for each page x in the base set:
 - a **hub score** $h(x)$
 - and an **authority score** $a(x)$.
- Initialize: for all x , $h(x) \leftarrow 1$; $a(x) \leftarrow 1$;
- Iteratively update all $h(x)$, $a(x)$;

Distilling hubs and authorities



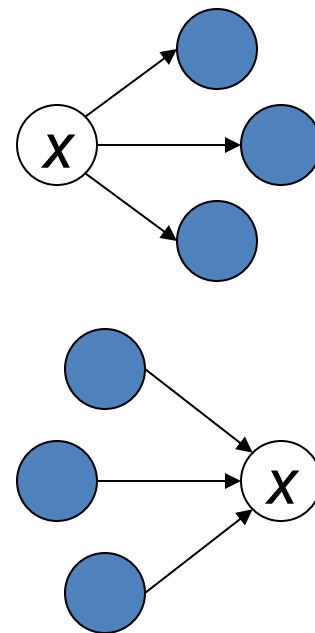
- Compute, for each page x in the base set:
 - a **hub score** $h(x)$
 - and an **authority score** $a(x)$.
- Initialize: for all x , $h(x) \leftarrow 1$; $a(x) \leftarrow 1$;
- Iteratively update all $h(x)$, $a(x)$;
- After a few iterations (or convergence)
 - output pages with highest $h()$ scores as top hubs
 - highest $a()$ scores as top authorities.

Iterative update

- Repeat the following updates of the **hub score $h(x)$** and the **authority score $a(x)$** , for all x :

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$

$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$



Scaling



- To prevent the $h()$ and $a()$ values from getting too big, can scale down after each iteration.

Scaling



- To prevent the $h()$ and $a()$ values from getting too big, can scale down after each iteration.
- *Scaling factor doesn't really matter (as long as you are consistent):*
 - *we only care about the relative values of the scores.*

How many iterations?



- Claim: relative values of scores will converge after a few iterations:
 - in fact, suitably scaled, $h()$ and $a()$ scores settle into a steady state!
 - *You can demonstrate this...*
- In practice, ~5 iterations is enough to get you close to stability.

Japan Elementary Schools



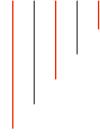
Hubs

- schools
 - LINK Page-13
 - “ú-{}íŠw□Z
 - □a‰o,,□¬Šw□Zfz□[f□fy□[fW
 - 100 Schools Home Pages (English)
 - K-12 from Japan 10/...rnet and Education)
 - http://www...iglobe.ne.jp/~IKESAN
 - ,l,f,j□¬Šw□Z,U”N,P'g•“Œê
 - □ÒŠ—’—§□ÒŠ—“Œ□¬Šw□Z
 - Koulutus ja oppilaitokset
 - TOYODA HOMEPAGE
 - Education
 - Cay's Homepage(Japanese)
 - –y“í□¬Šw□Z,ífz□[f□fy□[fW
 - UNIVERSITY
 - %oJ—³□¬Šw□Z DRAGON97-TOP
 - □Â‰o^□¬Šw□Z,T”N,P'gfz□[f□fy□[fW
 - ¶µºé¼ÂÁ© ¥á¥Ê¥å½ ¥á¥Ê¥å½

Authorities

- The American School in Japan
 - The Link Page
 - %o^è)s—§^ä“c□¬Šw□Zfz□[f□fy□[fW
 - Kids' Space
 - ^À□é□s—§^À□é□¼•”□¬Šw□Z
 - <{□é<³^ç'âŠw•□’®□¬Šw□Z
 - KEIMEI GAKUEN Home Page (Japanese)
 - Shiranuma Home Page
 - fuzoku-es.fukui-u.ac.jp
 - welcome to Miasa E&J school
 - □_“þ□íŒ\$□E%oj•|□s—§’†□í□¼□¬Šw□Z,¡fy
 - http://www...p/~m_maru/index.html
 - fukui haruyama-es HomePage
 - Torisu primary school
 - goo
 - Yakumo Elementary,Hokkaido,Japan
 - FUZOKU Home Page
 - Kamishibun Elementary School...

Things to note



- Pulled together good pages **regardless of language of page content.**

Things to note

- Pulled together good pages **regardless of language of page content.**
- Use *only* link analysis after base set assembled
 - **iterative scoring is query-independent.**

Things to note



- Pulled together good pages **regardless of language of page content.**
- Use *only* link analysis after base set assembled
 - **iterative scoring is query-independent.**
- Iterative computation after text index retrieval - significant overhead (but not at query-time!)

Hyper-link Induced Topic Search Issues



- **Topic Drift**
 - Off-topic pages can cause off-topic “authorities” to be returned
 - E.g., the neighborhood graph can be about a “super topic”

Hyper-link Induced Topic Search Issues

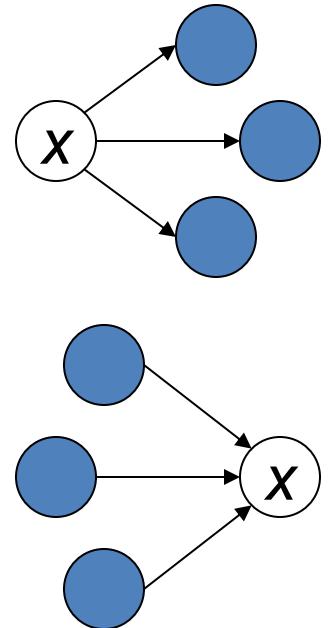


- **Topic Drift**
 - Off-topic pages can cause off-topic “authorities” to be returned
 - E.g., the neighborhood graph can be about a “super topic”
- **Mutually Reinforcing Affiliates**
 - Affiliated pages/sites can boost each others’ scores
 - *Linkage between affiliated pages is not a useful signal, so avoid considering them in the analysis*

Overview!

- Compute, for each page x in the base set:
 - a **hub score $h(x)$**
 - and an **authority score $a(x)$** .
- Initialize: for all x ,
$$h(x) \leftarrow 1; a(x) \leftarrow 1;$$
- Iteratively update all
$$h(x), a(x);$$
- After a few iterations (or convergence)
 - output pages with highest $h()$ scores as top hubs
 - highest $a()$ scores as top authorities.

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$



$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$

To prevent the $h()$ and $a()$ values from getting too big, can scale down after each iteration.

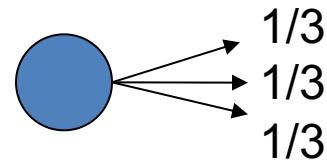
Scaling factor doesn't really matter : only relative values are important

PAGE RANK

Pagerank scoring

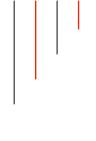


- Imagine a browser doing a random walk on web pages:
 - Start at a random page
 - At each step, go out of the current page along one of the links on that page, equiprobably

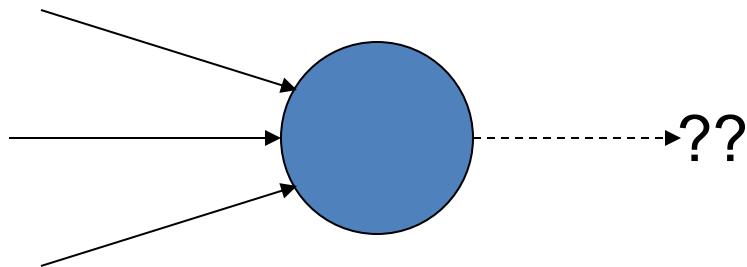


- “In the steady state” each page has a long-term visit rate - use this as the page’s score.

Not quite enough



- The web is full of dead-ends.
 - Random walk can get stuck in dead-ends.
 - Makes no sense to talk about long-term visit rates.



Teleporting

- At a dead end, jump to a random web page.
- At any non-dead end, with probability 10%, jump to a random web page.
 - With remaining probability (90%), go out on a random link.
 - 10% - a parameter.

Result of teleporting



- Now cannot get stuck locally.

Result of teleporting



- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited (not obvious).

Result of teleporting



- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited (not obvious).
 - **The idea of page rank is that pages that are visited more by this random walker are more important**

PageRank



- Status of searching the web in November 1997: “only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results).” Brin&Page, 1997

PageRank



- Status of searching the web in November 1997: “only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results).” Brin&Page, 1997
 - “Our main goal is to improve the quality of web search engines.”
Brin&Page, 1997

PageRank



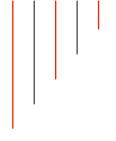
- Status of searching the web in November 1997: “only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results).” Brin&Page, 1997
 - “Our main goal is to improve the quality of web search engines.”
Brin&Page, 1997
- *See the original Google and PageRank papers in Canvas*

Result of teleporting



- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited (not obvious).
 - **The idea of page rank is that pages that are visited more by this random walker are more important**
- **How do we compute this visit rate?**

Computing PageRank



1. Algebraic calculation (given in Google paper)
2. Markov chain model

PageRank -- Algebraic



$$p_i = \boxed{\text{matemagic}} p_j$$

PageRank -- Algebraic



$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

PageRank -- Algebraic

$$\text{PageRank of page i} \quad p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j \quad \text{PageRank of page j}$$

PageRank -- Algebraic

PageRank of page i

$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

$L_{ij} = 1$ if page j points to i, and zero otherwise

PageRank of page j

$$c_j = \sum_{i=1}^N L_{ij}$$

How many pages are leaving page j

PageRank -- Algebraic

$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

PageRank of page i damping factor d PageRank of page j

$c_j = \sum_{i=1}^N L_{ij}$

L_{ij} = 1 if page j points to i, and zero otherwise

How many pages are leaving page j

The diagram illustrates the PageRank formula and its components. A red box encloses the formula for c_j . Red arrows point from the labels to their corresponding parts in the formula: 'damping factor d' to the term $(1 - d)$, 'PageRank of page i' to the first term, 'PageRank of page j' to the second term, 'How many pages are leaving page j' to the denominator c_j , and 'L_{ij} = 1 if page j points to i, and zero otherwise' to the term L_{ij} .

PageRank -- Algebraic



$$PR(A) = (1-d) + d \sum (PR(Ti)/C(Ti))$$

$$PR(A) = (1-d) + d (PR(T1)/C(T1) + \dots + PR(Tn)/C(Tn))$$

Where

A is a page

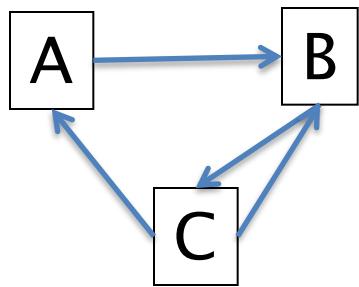
d is a damping/teleport factor (usually 0.85)

T1...Tn are pages that link to A

PR(Ti) is the PageRank of Ti

C(Ti) is the number of outgoing links from Ti

Example



$$PR(A) = 0.15 + 0.85 * PR(C)/2$$

$$PR(B) = 0.15 + 0.85 * (PR(A)/1 + PR(C)/2)$$

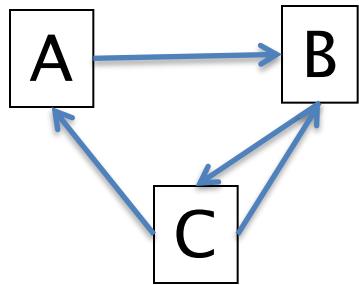
$$PR(C) = 0.15 + 0.85 * (PR(B)/1)$$

How do we start?!?

Guess: $PR(p) = 1$ for starters

Then, iterate

Example – Iterations

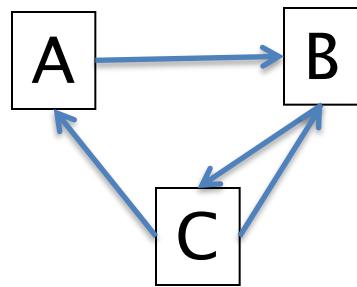


$$\begin{aligned} PR(A) &= 0.15 + 0.85 * PR(C)/2 \\ PR(B) &= 0.15 + 0.85 * (PR(A)/1 + PR(C)/2) \\ PR(C) &= 0.15 + 0.85 * (PR(B)/1) \end{aligned}$$

Iter 1 {

$$\begin{aligned} PR(A) &= 0.15+0.85 * 1/2 = 0.575 \\ PR(B) &= 0.15+0.85 * (1+1/2) = 1.425 \\ PR(C) &= 0.15+0.85 * 1 = 1 \end{aligned}$$

Example – Iterations



$$\begin{aligned} PR(A) &= 0.15 + 0.85 * PR(C)/2 \\ PR(B) &= 0.15 + 0.85 * (PR(A)/1 + PR(C)/2) \\ PR(C) &= 0.15 + 0.85 * (PR(B)/1) \end{aligned}$$

Iter 1 {

$$\begin{aligned} PR(A) &= 0.15+0.85 * 1/2 = 0.575 \\ PR(B) &= 0.15+0.85 * (1+1/2) = 1.425 \\ PR(C) &= 0.15+0.85 * 1 = 1 \end{aligned}$$

Iter 2 {

$$\begin{aligned} PR(A) &= 0.15+0.85 * 1/2 = 0.575 \\ PR(B) &= 0.15+0.85 * (0.575+1/2) = 1.06375 \\ PR(C) &= 0.15+0.85 * 1.425 = 1.36125 \end{aligned}$$

...

Example -- Iterations



- Algorithm converges after N iterations
- Once converged, normalized probability distribution (average PageRank for all pages) will be 1
- You can gain some visual intuition :
<http://edilogues.com/projects/2015/03/22/pagerank-visualization>

Pagerank summary



- **Preprocessing:**
 - Given graph of links, build a (sparse) matrix of the links.
 - From it compute **page rank**
- One possibility to add PR in query processing (at query time):
 - Retrieve pages meeting query.
 - Rank them by their pagerank.
 - But this rank order is *query-independent* ...

The reality



- Pagerank is used in google and other engines, but is hardly the full story of ranking
 - **It is just one among many criteria used in relevance scoring**
 - Many additional sophisticated features are used
 - Some address specific query classes
 - Machine learned ranking is used each time more

The reality

- Pagerank is used in google and other engines, but is hardly the full story of ranking
 - It is just one among many criteria used in relevance scoring
 - Many additional sophisticated features are used
 - Some address specific query classes
 - Machine learned ranking is used each time more
- **Given a query, a real-life search engine will compute a composite score for each web page that combines hundreds of features**
(cosine similarity, term proximity, PageRank, HITS, etc.)
- Pagerank *alone* is still very useful for things like crawl policy

PageRank – Algebraic in matrix notation

$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

PageRank – Algebraic in matrix notation

$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

$$\mathbf{p} = (1 - d)\mathbf{e} + d \cdot \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Vector of N ones

Diagonal matrix with diagonal elements c_j

PageRank – Algebraic in matrix notation

$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

$$\mathbf{p} = (1 - d)\mathbf{e} + d \cdot \mathbf{LD}_c^{-1}\mathbf{p}$$

$$\mathbf{e}^T \mathbf{p} = N$$

If we adopt a normalization factor
such that the average pagerank is 1

PageRank – Algebraic in matrix notation

$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

$$\mathbf{p} = (1 - d)\mathbf{e} + d \cdot \mathbf{LD}_c^{-1}\mathbf{p} \quad \mathbf{e}^T \mathbf{p} = N$$

$$\begin{aligned} \mathbf{p} &= \left[(1 - d)\mathbf{e}\mathbf{e}^T/N + d\mathbf{LD}_c^{-1} \right] \mathbf{p} \\ &= \mathbf{A}\mathbf{p} \end{aligned}$$

PageRank – Algebraic in matrix notation

$$p_i = (1 - d) + d \sum_{j=1}^N \left(\frac{L_{ij}}{c_j} \right) p_j$$

$$\mathbf{p} = (1 - d)\mathbf{e} + d \cdot \mathbf{LD}_c^{-1}\mathbf{p} \quad \mathbf{e}^T \mathbf{p} = N$$

$$\begin{aligned} \mathbf{p} &= \left[(1 - d)\mathbf{e}\mathbf{e}^T/N + d\mathbf{LD}_c^{-1} \right] \mathbf{p} \\ &= \mathbf{A}\mathbf{p} \end{aligned}$$

$$\mathbf{p}_k \leftarrow \mathbf{A}\mathbf{p}_{k-1}; \quad \mathbf{p}_k \leftarrow N \frac{\mathbf{p}_k}{\mathbf{e}^T \mathbf{p}_k}.$$

Informatics 225

Computer Science 221

Information Retrieval

Lecture 25

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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.

Search Engine Evaluation

Information Retrieval

Architecture

Preprocessing Steps

Index Creation



Index



Ranking

Text Transformation/Processing

Text Acquisition



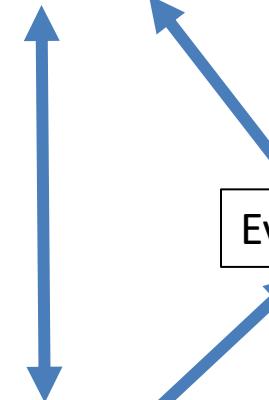
Local Document Store

Web Pages

Querying Process



UI



Evaluation

Log



Evaluation



- For a given query, a corpus and a specific definition of relevance:

Evaluation

- For a given query, a corpus and a specific definition of relevance:

Effectiveness and efficiency

- Effectiveness : measures the ability to find the right information
 - How well the rank corresponds to the rank that the user expects

Evaluation

- For a given query, a corpus and a specific definition of relevance:

Effectiveness and efficiency

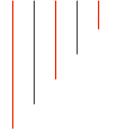
- Effectiveness : measures the ability to find the right information
 - How well the rank corresponds to the rank that the user expects
- Efficiency : measures how quickly this is done, how many resources are needed for this to be done
 - Time and space requirements of the methods that produced the ranking

Evaluation



- Evaluation is **key** to building *effective* and *efficient* search engines
 - measurement usually carried out in **controlled laboratory experiments**
 - *online* testing can also be done

Evaluation



- Evaluation is key to building *effective* and *efficient* search engines
 - measurement usually carried out in controlled laboratory experiments
 - *online* testing can also be done
- Effectiveness, efficiency and cost are related
 - e.g., if we want a particular level of effectiveness and efficiency, this will determine the cost of the system configuration
 - efficiency and cost targets may impact effectiveness

Evaluation Corpus

- Allow results from different methods to be compared



Evaluation Corpus

- Allow results from different methods to be compared
- *Test collections* consisting of documents, queries, and relevance judgments, e.g.,
 - CACM: Titles and abstracts from the Communications of the ACM from 1958-1979. Queries and relevance judgments generated by computer scientists.

Evaluation Corpus

- Allow results from different methods to be compared
- *Test collections* consisting of documents, queries, and relevance judgments, e.g.,
 - CACM: Titles and abstracts from the Communications of the ACM from 1958-1979. Queries and relevance judgments generated by computer scientists.
 - AP: Associated Press newswire documents from 1988-1990 (from TREC disks 1-3). Queries are the title fields from TREC topics 51-150. Topics and relevance judgments generated by government information analysts.
 - GOV2: Web pages crawled from websites in the .gov domain during early 2004. Queries are the title fields from TREC topics 701-850. Topics and relevance judgments generated by government analysts.

Test Collections

Collection	Number of documents	Size	Average number of words/doc.
CACM	3,204	2.2 Mb	64
AP	242,918	0.7 Gb	474
GOV2	25,205,179	426 Gb	1073

Collection	Number of queries	Average number of words/query	Average number of relevant docs/query
CACM	64	13.0	16
AP	100	4.3	220
GOV2	150	3.1	180

TREC Topic Example for a query

<top>
<num> Number: 794

<title> pet therapy  The title is used as query

<desc> Description:
How are pets or animals used in therapy for humans and what are the benefits?

<narr> Narrative:
Relevant documents must include details of how pet- or animal-assisted therapy is or has been used. Relevant details include information about pet therapy programs, descriptions of the circumstances in which pet therapy is used, the benefits of this type of therapy, the degree of success of this therapy, and any laws or regulations governing it.

</top>

Relevance Judgments



- Obtaining relevance judgments is an expensive, time-consuming process
 - who does it?
 - what are the instructions?
 - what is the level of agreement?

Relevance Judgments



- Obtaining relevance judgments is an expensive, time-consuming process
 - who does it?
 - what are the instructions?
 - what is the level of agreement?
- TREC judgments
 - depend on task being evaluated
 - generally binary
 - agreement good because of “narrative”

Pooling



- Exhaustive judgments for all documents in a collection is not practical

Pooling



- Exhaustive judgments for all documents in a collection is not practical
- Pooling technique is used in GOV2
 - top k results (*for TREC, k varied between 50 and 200*) from the rankings obtained by different search engines (or retrieval algorithms) are merged into a pool
 - duplicates are removed
 - documents are presented in some random order to the relevance judges

Pooling



- Exhaustive judgments for all documents in a collection is not practical
- Pooling technique is used in GOV2
 - top k results (*for TREC, k varied between 50 and 200*) from the rankings obtained by different search engines (or retrieval algorithms) are merged into a pool
 - duplicates are removed
 - documents are presented in some random order to the relevance judges
- Produces a large number of relevance judgments for each query, although still incomplete
 - Studies have shown that comparisons are accurate



Query Logs

- Used for both tuning and evaluating search engines
 - also for various techniques such as query suggestion

Query Logs



- Used for both tuning and evaluating search engines
 - also for various techniques such as query suggestion
- Can generate privacy concerns
 - Anonymize the logged data?

Query Logs

- Used for both tuning and evaluating search engines
 - also for various techniques such as query suggestion
- Can generate privacy concerns
 - Anonymize the logged data?
- Typical contents
 - User identifier or user session identifier
 - Query terms - stored exactly as user entered
 - List of URLs of results, their ranks on the result list, and whether they were clicked on
 - Timestamp(s) - records the time of user events such as query submission, clicks
 - In which page the user Clicked!!!

Query Logs



- **Clicks are not relevance judgments**
 - although they are correlated
 - biased by a number of factors such as rank on result list:
confirmation bias!

Query Logs



- Clicks are not relevance judgments
 - although they are correlated
 - biased by a number of factors such as rank on result list:
confirmation bias!
- Can use clickthrough data to predict *preferences* between pairs of documents
 - Correlated with relevance judgments
 - Appropriate for tasks with multiple levels of relevance, focused on user relevance
 - Various “policies” used to generate preferences

Example Click Policy



- *Skip Above and Skip Next* (Agichtein et al., 2006)

- Result document list and click data

d_1
 d_2
 d_3 (clicked)
 d_4

- Generated preferences

$d_3 > d_2$
 $d_3 > d_1$
 $d_3 > d_4$

Query Logs

- Click data of a single user can be noisy!
 - But click data can also be aggregated to remove noise

Query Logs



- Click data of a single user can be noisy!
 - But click data can also be aggregated to remove noise
- *Click distribution* information
 - can be used to identify clicks that have a higher frequency than would be expected
 - high correlation with relevance
 - e.g., using *click deviation* to filter clicks for preference-generation policies

Filtering Clicks



- *Click deviation $CD(d, p)$* for a result d in position p :

$$CD(d, p) = O(d, p) - E(p)$$

$O(d, p)$: observed click frequency for a document in a rank position p over
all instances of a given query

$E(p)$: expected click frequency at rank p *averaged across all queries*

- Use this value to filter clicks and provide more reliable information (optimizing your ranking).

Effectiveness Measures

A is set of relevant documents,
 B is set of retrieved documents

Recall : how well the search engine is doing at finding all the relevant documents for a query.

Precision : how well it is doing at rejecting non-relevant documents.

Effectiveness Measures

A is set of relevant documents,
 B is set of retrieved documents

	Relevant	Non-Relevant
Retrieved	$A \cap B$	$\overline{A} \cap B$
Not Retrieved	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$

$$Recall = \frac{|A \cap B|}{|A|}$$

$$Precision = \frac{|A \cap B|}{|B|}$$

Recall : how well the search engine is doing at finding all the relevant documents for a query.

Precision : how well it is doing at rejecting non-relevant documents.

Classification Errors



- *False Positive* (Type I error)
 - a non-relevant document is retrieved

$$Fallout = \frac{|\overline{A} \cap B|}{|\overline{A}|}$$

Classification Errors



- *False Positive* (Type I error)
 - a non-relevant document is retrieved

$$Fallout = \frac{|\overline{A} \cap B|}{|\overline{A}|}$$

- *False Negative* (Type II error)
 - Relevant documents that are not retrieved
 - $1 - Recall$

Informatics 225

Computer Science 221

Information Retrieval

Lecture 26

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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.

Search Engine Evaluation

Information Retrieval

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

Architecture

Preprocessing Steps

Index Creation



Index



Ranking

Text Transformation/Processing

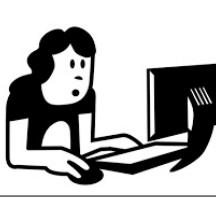
Text Acquisition



Local Document Store

Web Pages

Querying Process



UI

Evaluation



Evaluation

Log



Effectiveness Measures

A is set of relevant documents,
 B is set of retrieved documents

	Relevant	Non-Relevant
Retrieved	$A \cap B$	$\overline{A} \cap B$
Not Retrieved	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$

$$Recall = \frac{|A \cap B|}{|A|}$$

$$Precision = \frac{|A \cap B|}{|B|}$$

Recall : how well the search engine is doing at finding all the relevant documents for a query.

Precision : how well it is doing at rejecting non-relevant documents.

Classification Errors



- *False Positive* (Type I error)
 - a non-relevant document is retrieved

$$Fallout = \frac{|\overline{A} \cap B|}{|\overline{A}|}$$

- *False Negative* (Type II error)
 - Relevant documents that are not retrieved
 - $1 - Recall$

F Measure



- *Is there a single metric that could merge those values?*

F Measure

- *Is there a single metric that could merge those values?*
- *Harmonic mean* of recall and precision

$$F = \frac{1}{\frac{1}{2}(\frac{1}{R} + \frac{1}{P})} = \frac{2RP}{(R+P)}$$

- harmonic mean **emphasizes the importance of small values**, whereas the arithmetic mean is more affected by **outliers** that are unusually large

F Measure

- *Is there a single metric that could merge those values?*
- *Harmonic mean* of recall and precision

$$F = \frac{1}{\frac{1}{2}(\frac{1}{R} + \frac{1}{P})} = \frac{2RP}{(R+P)}$$

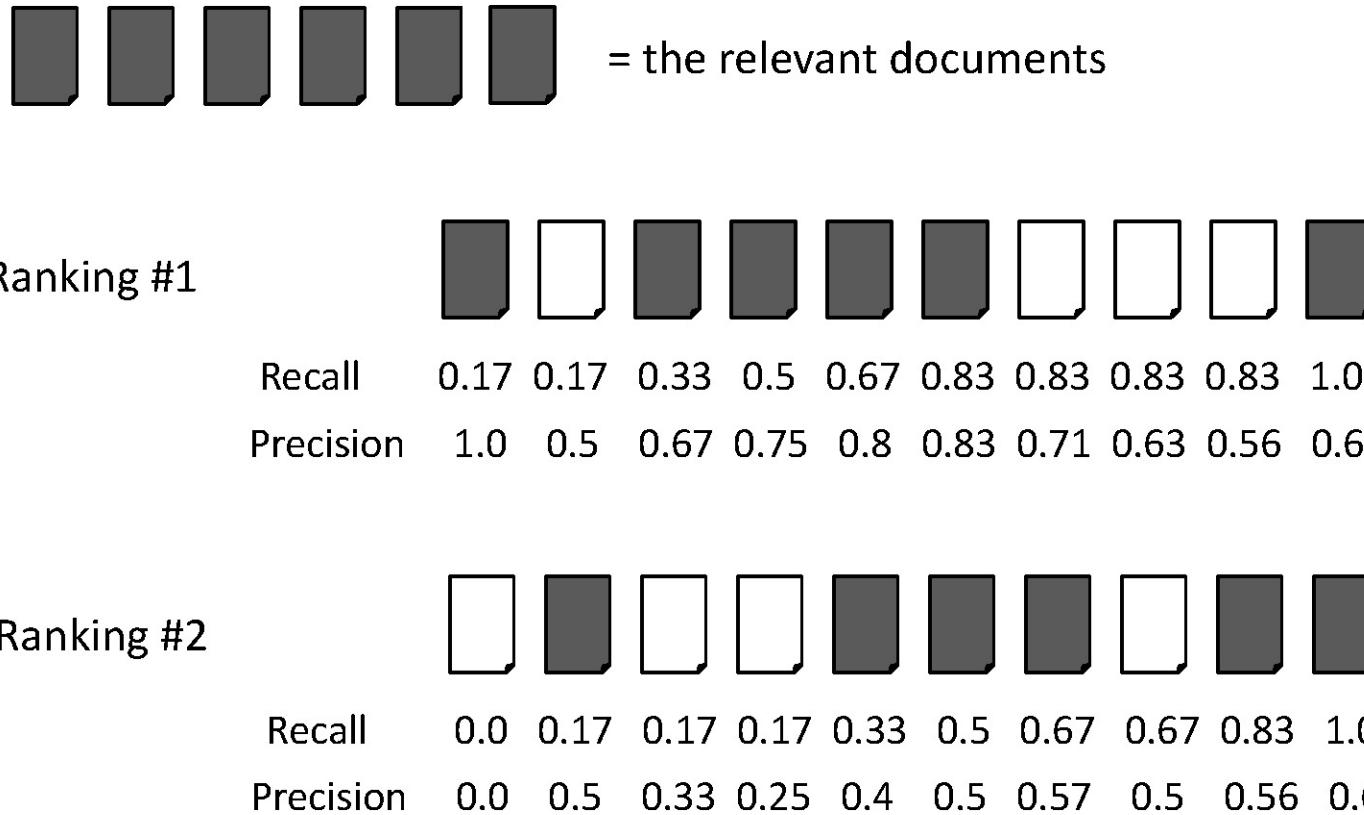
- harmonic mean emphasizes the importance of small values, whereas the arithmetic mean is more affected by outliers that are unusually large
- More general form (weighted)
$$F_\beta = (\beta^2 + 1)RP / (R + \beta^2 P)$$
 - β is a parameter that determines relative importance of recall and precision



- Set a rank threshold K
- Compute % relevant in top K
- Ignore documents ranked lower than K
- Ex:
 - Prec@3 of 2/3
 - Prec@4 of 2/4
 - Prec@5 of 3/5
- In similar fashion we have Recall@K



Ranking Effectiveness



Summarizing a Ranking

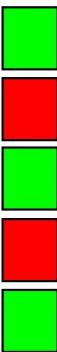


- You can:
 - Calculating recall and precision at **fixed rank positions**
 - Calculating precision at standard recall levels, from 0.0 to 1.0
 - requires *interpolation*
 - **Averaging** the precision values from the rank positions where a relevant document was retrieved



Average Precision

- Consider rank position of each **relevant** doc
 - $K_1, K_2, \dots K_R$
- Compute Precision@K for each $K_1, K_2, \dots K_R$
- **Average precision = average of P@K**

- Ex:  has AvgPrec of $\frac{1}{3} \cdot \left(\frac{1}{1} + \frac{2}{3} + \frac{3}{5} \right) \approx 0.76$

Average Precision

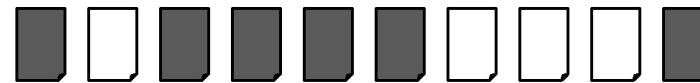


QUESTION



= the relevant documents

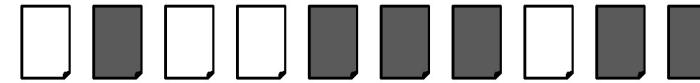
Ranking #1



Recall 0.17 0.17 0.33 0.5 0.67 0.83 0.83 0.83 0.83 1.0

Precision 1.0 0.5 0.67 0.75 0.8 0.83 0.71 0.63 0.56 0.6

Ranking #2



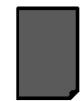
Recall 0.0 0.17 0.17 0.17 0.33 0.5 0.67 0.67 0.83 1.0

Precision 0.0 0.5 0.33 0.25 0.4 0.5 0.57 0.5 0.56 0.6

$$\text{Ranking } \#1: (1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6) / 6 = 0.78$$

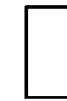
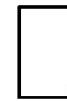
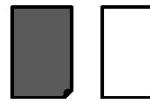
$$\text{Ranking } \#2: (0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6) / 6 = 0.52$$

Averaging Across Queries



= relevant documents for query 1

Ranking #1



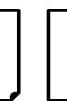
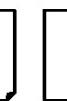
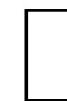
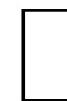
	Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
--	--------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

	Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5
--	-----------	-----	-----	------	-----	-----	-----	------	------	------	-----



= relevant documents for query 2

Ranking #2



	Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
--	--------	-----	------	------	------	------	------	-----	-----	-----	-----

	Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3
--	-----------	-----	-----	------	------	-----	------	------	------	------	-----



Averaging

- *Mean Average Precision (MAP)*
 - summarize rankings from multiple queries by averaging average precision
 - most commonly used measure in research papers

Averaging



- *Mean Average Precision (MAP)*
 - summarize rankings from multiple queries by averaging average precision
 - most commonly used measure in research papers
 - assumes user is interested in finding many relevant documents for each query
 - requires many relevance judgments in text collection



= relevant documents for query 1

Ranking #1									
Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44

= relevant documents for query 2

Ranking #2									
Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33

$$\text{average precision query 1} = (1.0 + 0.67 + 0.5 + 0.44 + 0.5) / 5 = 0.62$$
$$\text{average precision query 2} = (0.5 + 0.4 + 0.43) / 3 = 0.44$$

$$\text{mean average precision} = (0.62 + 0.44) / 2 = 0.53$$

Averaging



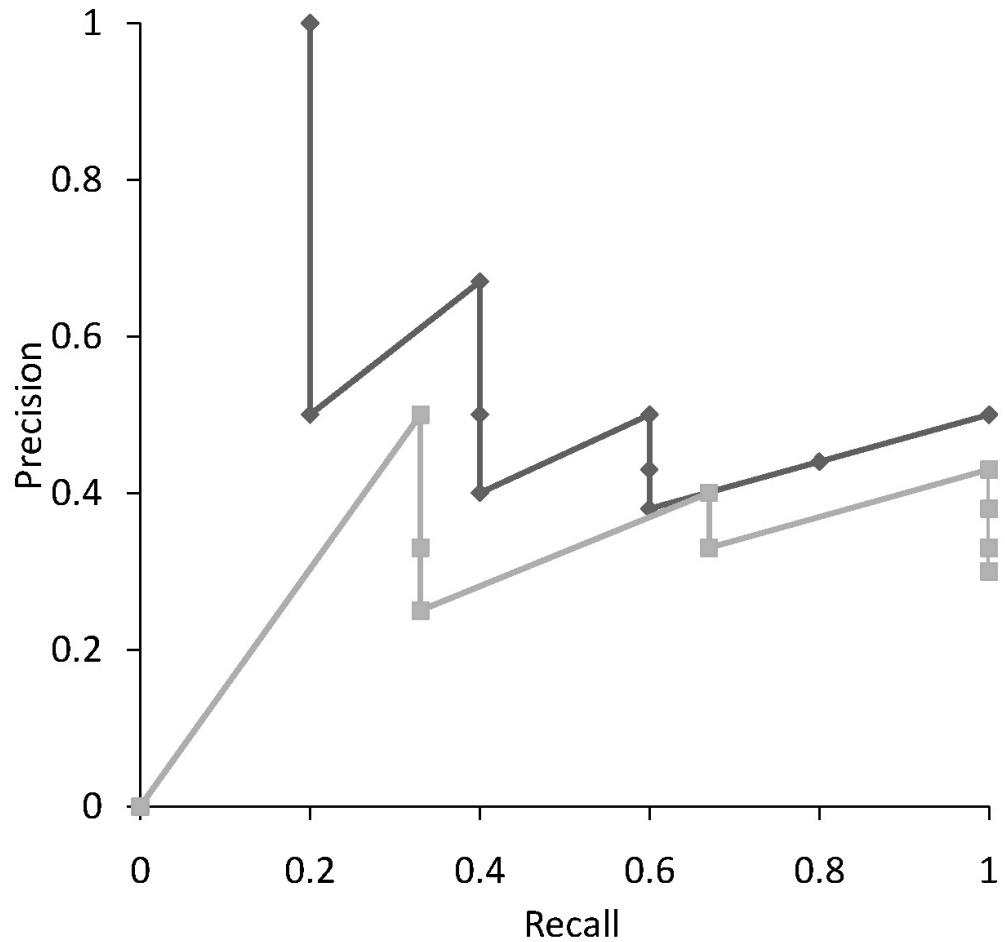
- *Mean Average Precision (MAP)*
 - summarize rankings from multiple queries by averaging average precision
 - most commonly used measure in research papers
 - assumes user is interested in finding many relevant documents for each query
 - requires many relevance judgments in text collection
- Recall-precision graphs are also useful summaries

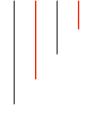
Summarizing a Ranking



- You can:
 - Calculating recall and precision at **fixed rank positions**
 - Calculating precision at standard recall levels, from 0.0 to 1.0
 - requires *interpolation*
 - **Averaging** the precision values from the rank positions where a relevant document was retrieved

Recall-Precision Graph





Interpolation

- Calculate precision at standard recall levels:

$$P(R) = \max\{P' : R' \geq R \wedge (R', P') \in S\}$$

Interpolation

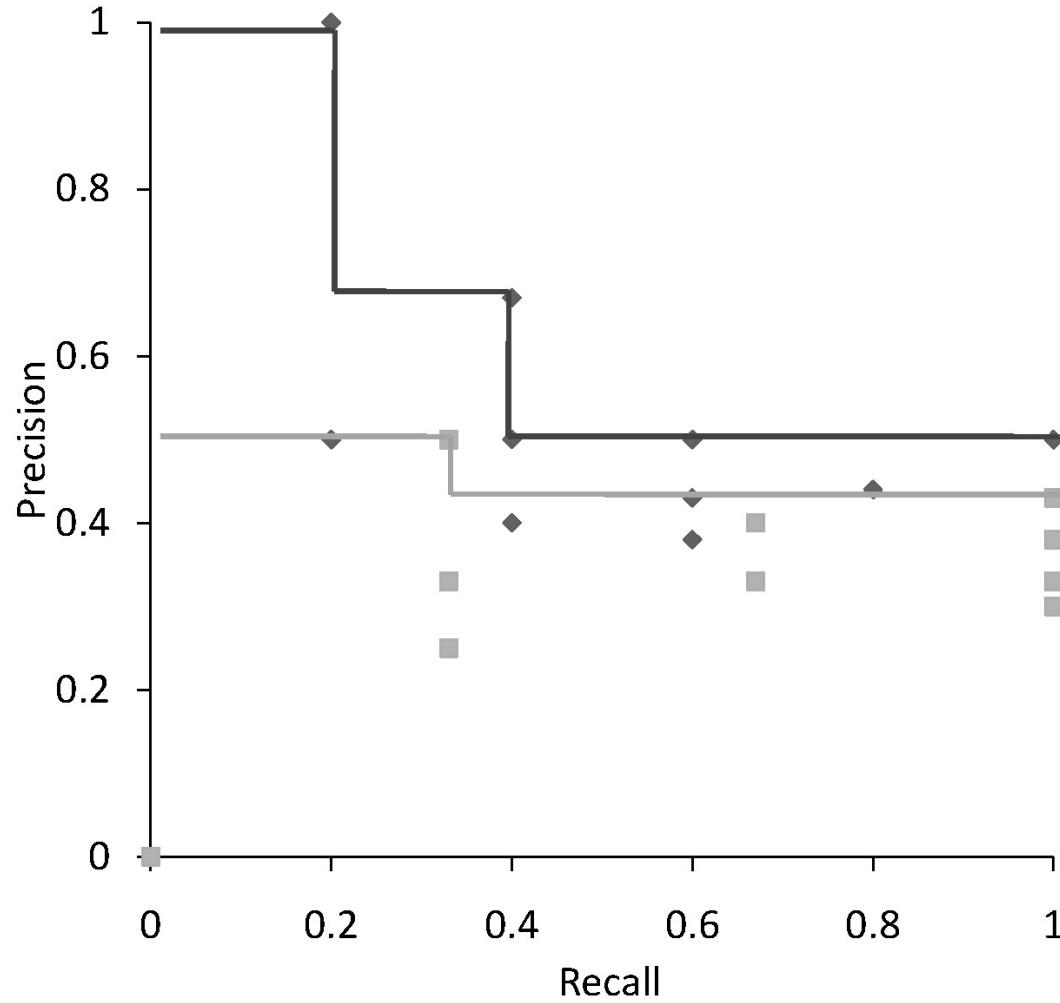


- Calculate precision at standard recall levels:

$$P(R) = \max\{P' : R' \geq R \wedge (R', P') \in S\}$$

- where S is the set of observed (R, P) points
- Defines precision at any recall level as the *maximum* precision observed in any recall-precision point at a higher recall level
 - produces a step function
 - defines precision at recall 0.0

Interpolation

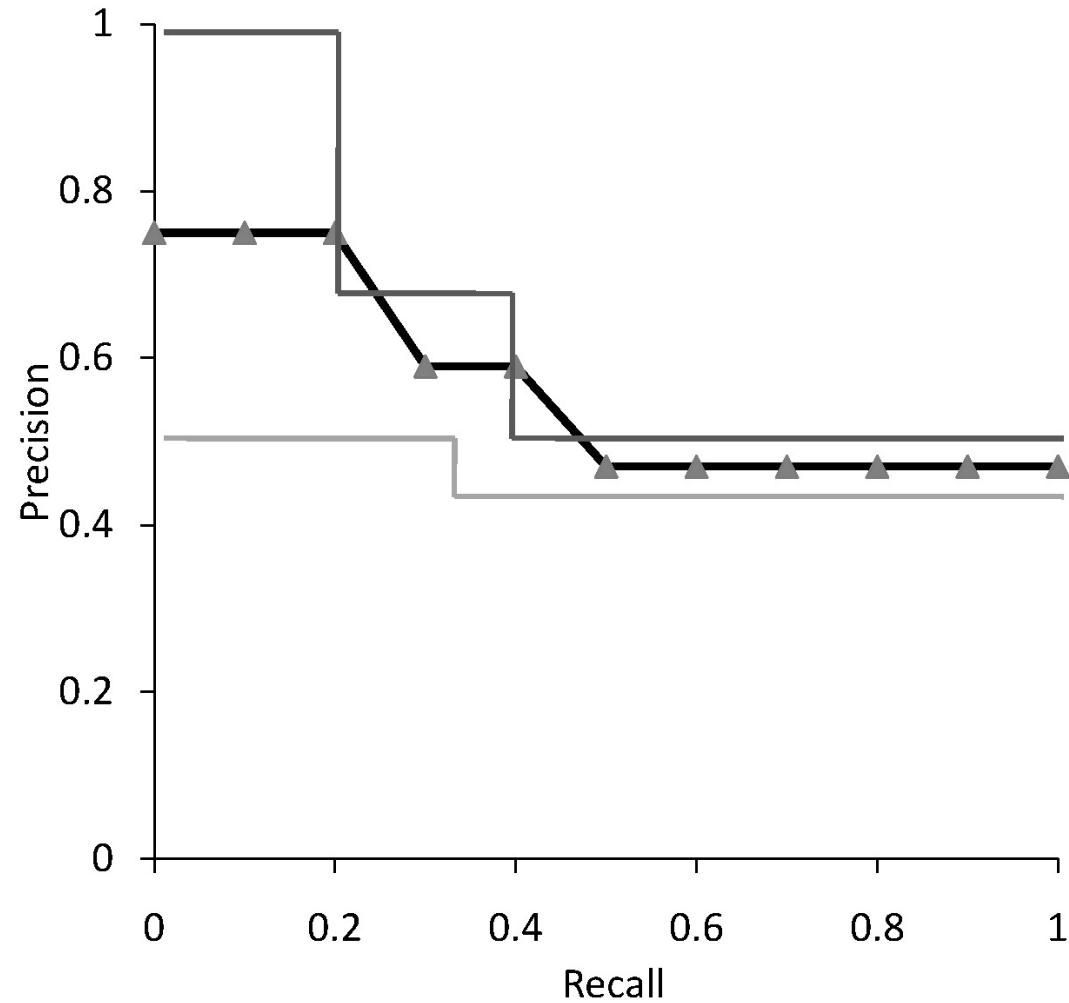


Average Precision at Standard Recall Levels

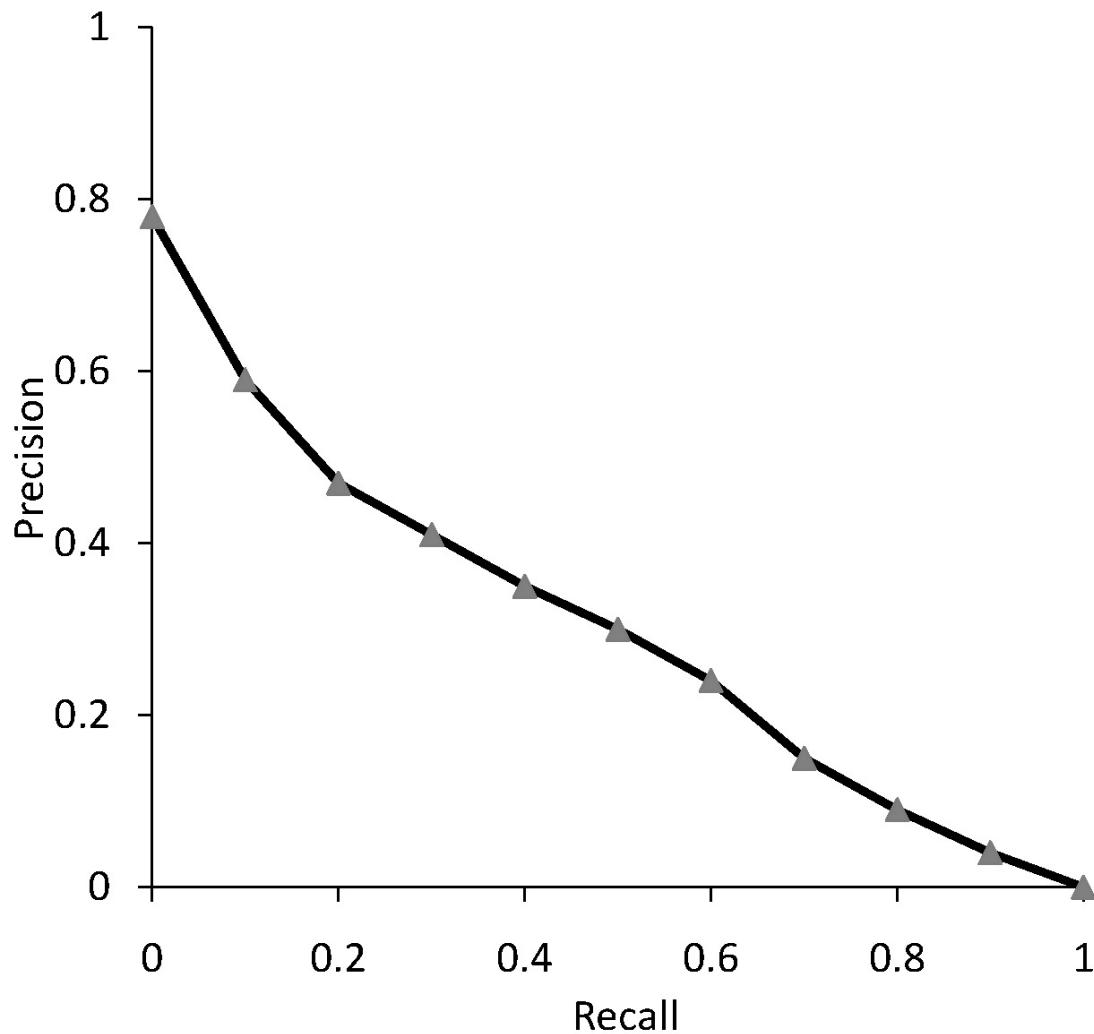
Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Ranking 1	1.0	1.0	1.0	0.67	0.67	0.5	0.5	0.5	0.5	0.5	0.5
Ranking 2	0.5	0.5	0.5	0.5	0.43	0.43	0.43	0.43	0.43	0.43	0.43
Average	0.75	0.75	0.75	0.59	0.47	0.47	0.47	0.47	0.47	0.47	0.47

- Recall-precision graph plotted by simply joining the average precision points at the standard recall levels

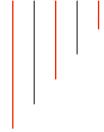
Average Recall-Precision Graph



Graph for 50 Queries



Focusing on Top Documents



- Users tend to look at only the top part of the ranked result list to find relevant documents
- Some search tasks have only one relevant document
 - e.g., navigational search, question answering
- Recall not always appropriate
 - instead need to measure how well the search engine does at retrieving relevant documents at very high ranks

Focusing on Top Documents



- Precision at Rank R
 - R typically 5, 10, 20
 - easy to compute, average, understand
 - not sensitive to rank positions less than R

Focusing on Top Documents



- Precision at Rank R
 - R typically 5, 10, 20
 - easy to compute, average, understand
 - not sensitive to rank positions less than R
- Reciprocal Rank
 - reciprocal of the rank at which the first relevant document is retrieved
 - *Mean Reciprocal Rank (MRR)* is the average of the reciprocal ranks over a set of queries
 - very sensitive to rank position

Discounted Cumulative Gain

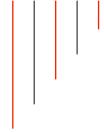
- Popular measure for evaluating web search and related tasks
- Two (very reasonable) assumptions:
 - Highly relevant documents are more useful than marginally relevant documents
 - The lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined

Discounted Cumulative Gain



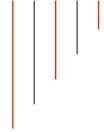
- Uses *graded relevance* as a measure of the **usefulness**, or *gain*, from examining a document

Discounted Cumulative Gain



- Uses *graded relevance* as a measure of the usefulness, or *gain*, from examining a document
- Gain is accumulated starting at the top of the ranking and may be reduced, or *discounted*, at lower ranks

Discounted Cumulative Gain



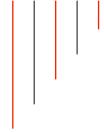
- Uses *graded relevance* as a measure of the usefulness, or *gain*, from examining a document
- Gain is accumulated starting at the top of the ranking and may be reduced, or *discounted*, at lower ranks
- Typical discount is $1/\log(\text{rank})$
 - With base 2, the discount at rank 4 is $1/2$, and at rank 8 it is $1/3$

Discounted Cumulative Gain

- What if relevance judgments are in a scale of $[0, r]$? $r > 2$
- **Cumulative Gain (CG) at rank n**
 - Let the ratings of the n documents be r_1, r_2, \dots, r_n (in ranked order)

$$CG = r_1 + r_2 + \dots + r_n$$

Discounted Cumulative Gain



- What if relevance judgments are in a scale of $[0, r]$? $r > 2$
- Cumulative Gain (CG) at rank n
 - Let the ratings of the n documents be r_1, r_2, \dots, r_n (in ranked order)

$$CG = r_1 + r_2 + \dots + r_n$$

- Discounted Cumulative Gain (DCG) at rank n

$$DCG = r_1 + r_2 / \log_2 2 + r_3 / \log_2 3 + \dots + r_n / \log_2 n$$

- *We may use any base for the logarithm!*

Discounted Cumulative Gain

- DCG is the total gain accumulated at a particular rank p :

$$DCG_p = rel_1 + \sum_{i=2}^p \frac{rel_i}{\log_2 i}$$

- Alternative formulation:

$$DCG_p = \sum_{i=1}^p \frac{2^{rel_i} - 1}{\log(1+i)}$$

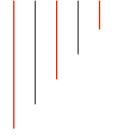
- used by some web search companies
- emphasis on retrieving highly relevant documents

DCG Example



- 10 ranked documents judged on 0-3 relevance scale:
 $3, 2, 3, 0, 0, 1, 2, 2, 3, 0$
- discounted gain:
 $3, 2/1, 3/1.59, 0, 0, 1/2.59, 2/2.81, 2/3, 3/3.17, 0$
 $= 3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0$
- DCG (*just add the discounted gains cumulatively!*):
 $3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61$

Normalized DCG



- DCG numbers are averaged across a set of queries at specific rank values
 - e.g., DCG at rank 5 is 6.89 and at rank 10 is 9.61
- DCG values are often *normalized* by comparing the DCG at each rank with the DCG value for the *perfect ranking*
 - makes averaging easier for queries with different numbers of relevant documents

NDCG Example

- Perfect ranking:

3, 3, 3, 2, 2, 2, 1, 0, 0, 0

- ideal DCG values:

3, 6, 7.89, 8.89, 9.75, 10.52, 10.88, 10.88, 10.88, 10

- NDCG values (*divide actual by ideal*):

1, 0.83, 0.87, 0.76, 0.71, 0.69, 0.73, 0.8, 0.88, 0.88

- $\text{NDCG} \leq 1$ at any rank position



Summary

- No single measure is the correct one for any application
 - choose measures appropriate for task
 - use a combination
 - shows different aspects of the system effectiveness

Summary



- No single measure is the correct one for any application
 - choose measures appropriate for task
 - use a combination
 - shows different aspects of the system effectiveness
- Analyze performance of individual queries
- Optimize the engine using user behavior (e.g. click logs)



Note: size of click logs

- How large is the click log?
 - **bing** search logs: 10+ TB/day
 - In existing publications:
 - [Silverstein+99]: 285M sessions
 - [Craswell+08]: 108k sessions
 - [Dupret+08] : 4.5M sessions (21 subsets * 216k sessions)
 - [Guo +09a] : 8.8M sessions from 110k unique queries
 - [Chapelle+09]: 58M sessions from 682k unique queries
 - [Liu+09a]: 0.26PB data from 103M unique queries

Informatics 225

Computer Science 221

Information Retrieval

Lecture 27

*Duplication of course material for any commercial purpose without
the explicit written permission of the professor is prohibited.*

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.

Introduction to Text classification

Information Retrieval

Categorization/Classification



- Given:
 - A representation of a document d
 - Issue: how to represent text documents.
 - Usually some type of high-dimensional space – bag of words
 - A fixed set of classes:
$$C = \{c_1, c_2, \dots, c_J\}$$
- Determine:
 - The category of d : $\gamma(d) \in C$, where $\gamma(d)$ is a classification function
 - We want to build classification functions (“classifiers”).

Supervised Learning



- Given:
 - A document d
 - A fixed set of classes:
$$C = \{c_1, c_2, \dots, c_J\}$$
 - A training set D of documents each with a label in C

Supervised Learning



- Given:
 - A document d
 - A fixed set of classes:
$$C = \{c_1, c_2, \dots, c_J\}$$
 - A training set D of documents each with a label in C
- Determine:
 - A learning method or algorithm which will enable us to learn a classifier γ
 - For a new test document dt , we assign it the class

$$\gamma(dt) \in C$$

Supervised Learning classification methods

- A few methods that are used for text
 - Naive Bayes (simple & common)
 - k-Nearest Neighbors (simple & powerful)
 - Support-vector machines (generally more powerful)
 - Decision trees → random forests →
gradient-boosted decision trees (e.g., xgboost)
 - Neural networks
 - ... plus many other methods
- *Supervised learning: need hand-classified training data*
- Many research and commercial systems use a mix of methods
 - Ensemble methods

Features



- Classifiers can use any sort of feature
 - URL, email address, punctuation, capitalization, dictionaries, network features
- In the simplest possible view: bag of words of documents
 - We use **only** word features
 - We use **all** of the words in the text (not a subset)

The bag of words representation

Y(

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet.

) = C

The bag of words representation

$\gamma(\quad) = c$

great	2
love	2
recommend	1
laugh	1
happy	1
...	...

Feature Selection: Why?

- Text collections have a large number of features
 - 10,000 – 1,000,000 unique words ... and more
- Selection may make a particular classifier feasible
 - Most classifiers can't deal with 1,000,000 features
(the curse of dimensionality)
- Reduces training time
 - Training time for some methods is quadratic or worse in number of features
- Makes runtime models smaller and faster
- Can improve generalization (performance)
 - Eliminates noisy features
 - Avoids overfitting

Feature Selection: Frequency of words?

- The simplest feature selection method:
 - Just use the commonest terms for a certain frequency threshold
 - No particular foundation
 - But it make sense why this works
 - They're the words that can be well-estimated and are most often available as evidence
 - In practice, this is ~90% as good as better methods (*but mileage may vary as this ~90% is averaged over all possible classes*)

Remember: Vector Space Representation



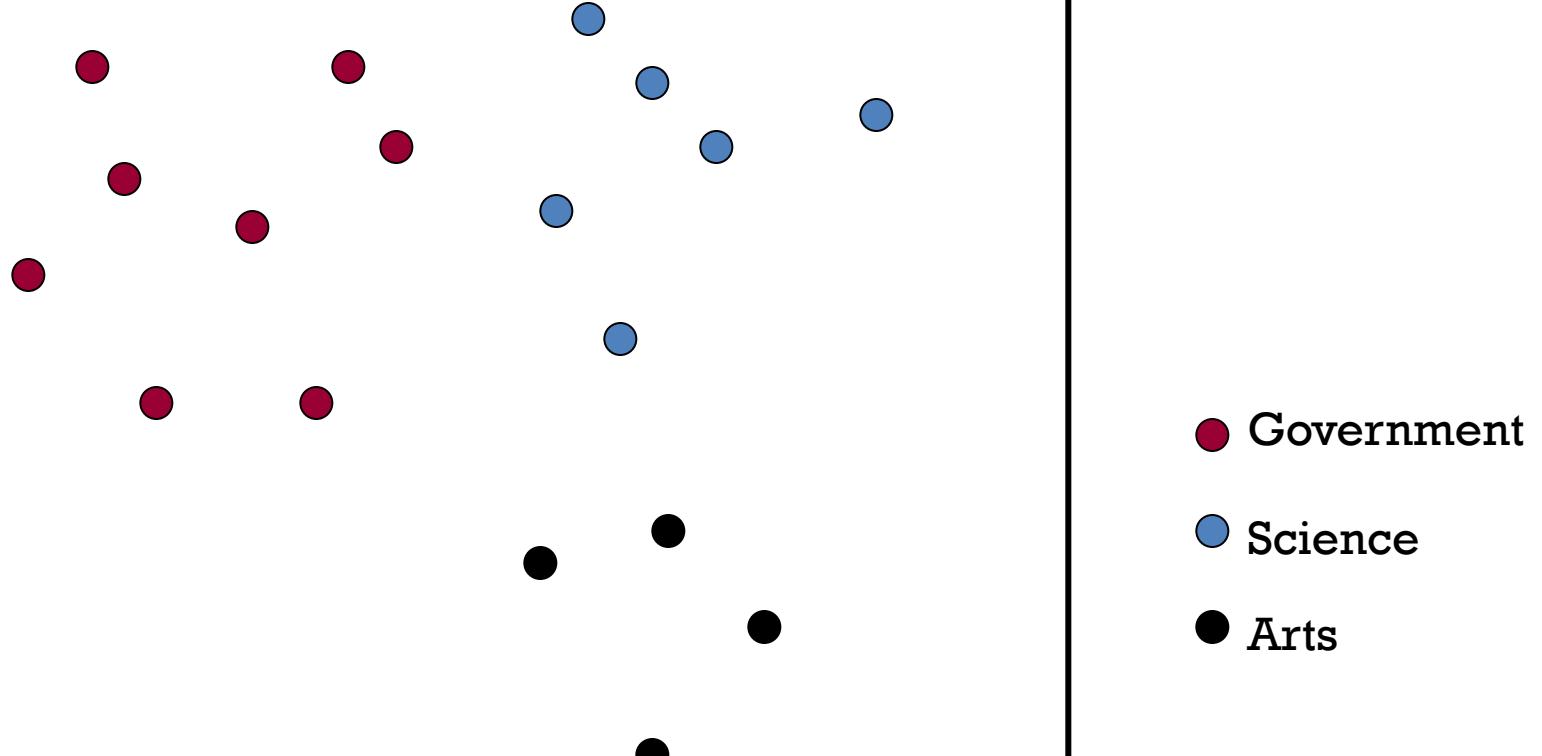
- Each document is a vector,
one component for each term (= word).
- Normally normalize vectors to unit length.
- High-dimensional vector space:
 - Terms are axes
 - 10,000+ dimensions, or even 100,000+
 - Docs are vectors in this space
- How can we do classification in this space?

Classification Using Vector Spaces



- In vector space classification, training set corresponds to a labeled set of points (equivalently, vectors)
- Premise 1: Documents in the same class form a contiguous region of space
- Premise 2: Documents from different classes don't overlap (much)
- Learning a classifier:
build surfaces to delineate classes in the space

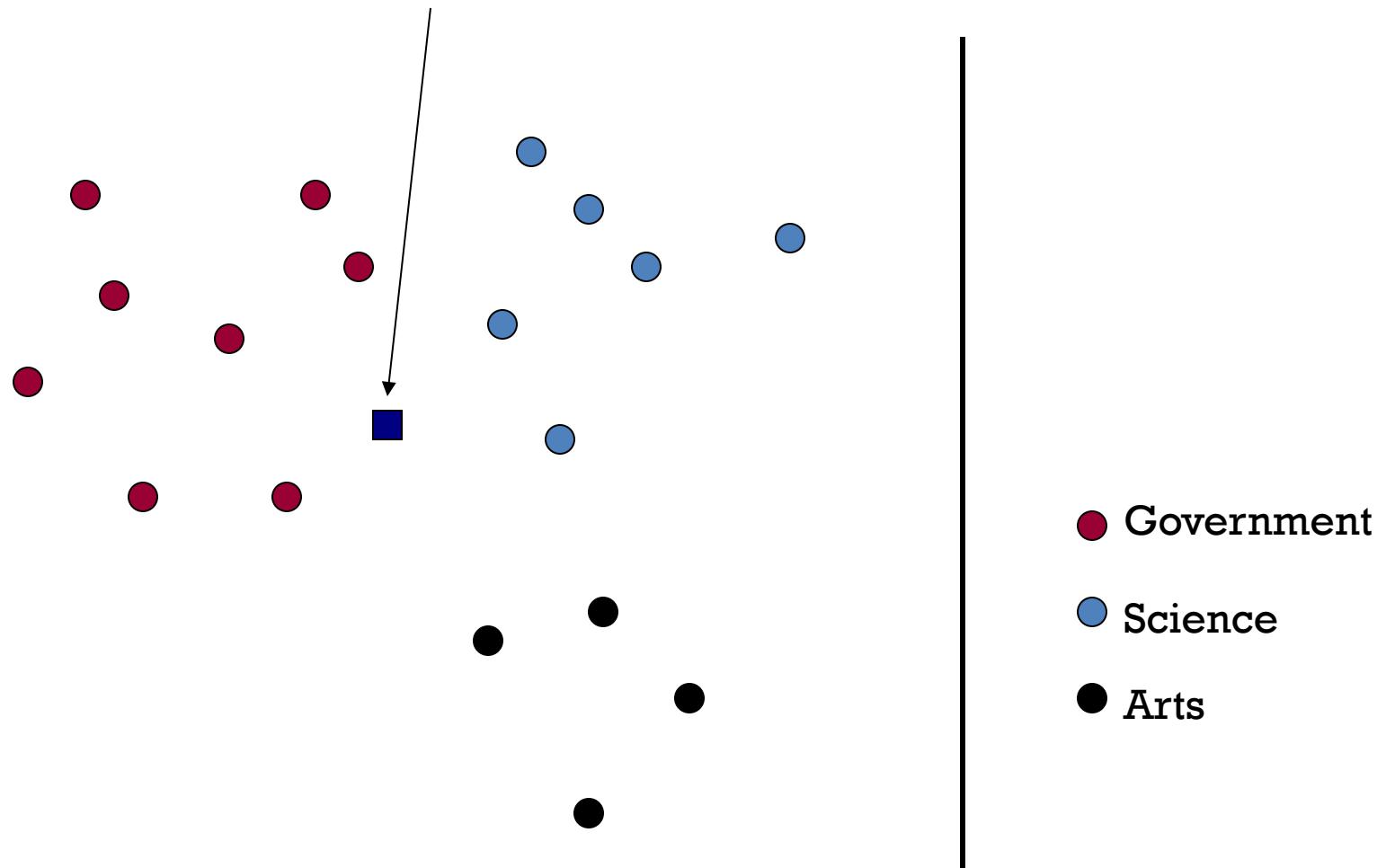
Documents in a Vector Space



Documents in a Vector Space



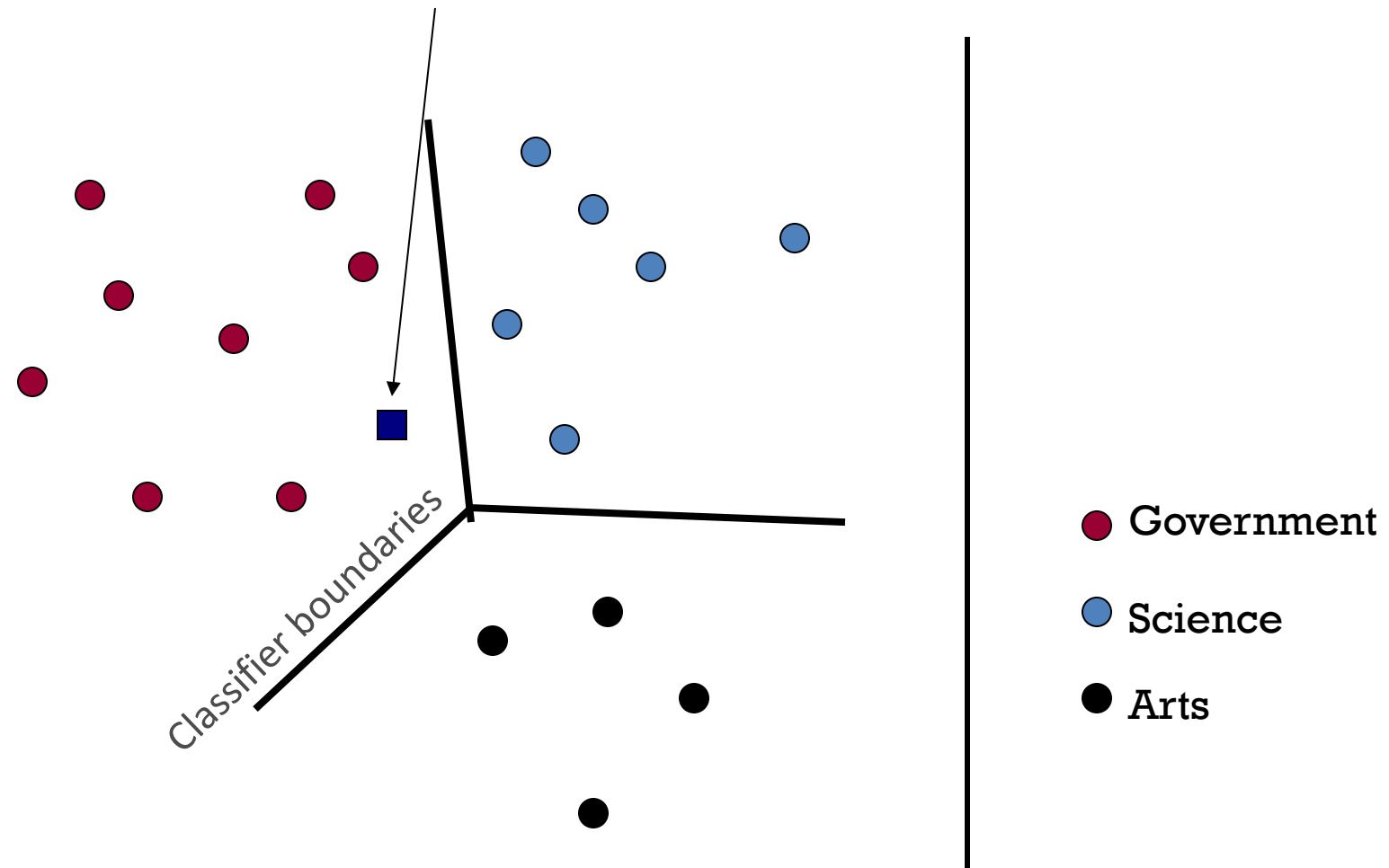
What is the class of this document?



Documents in a Vector Space



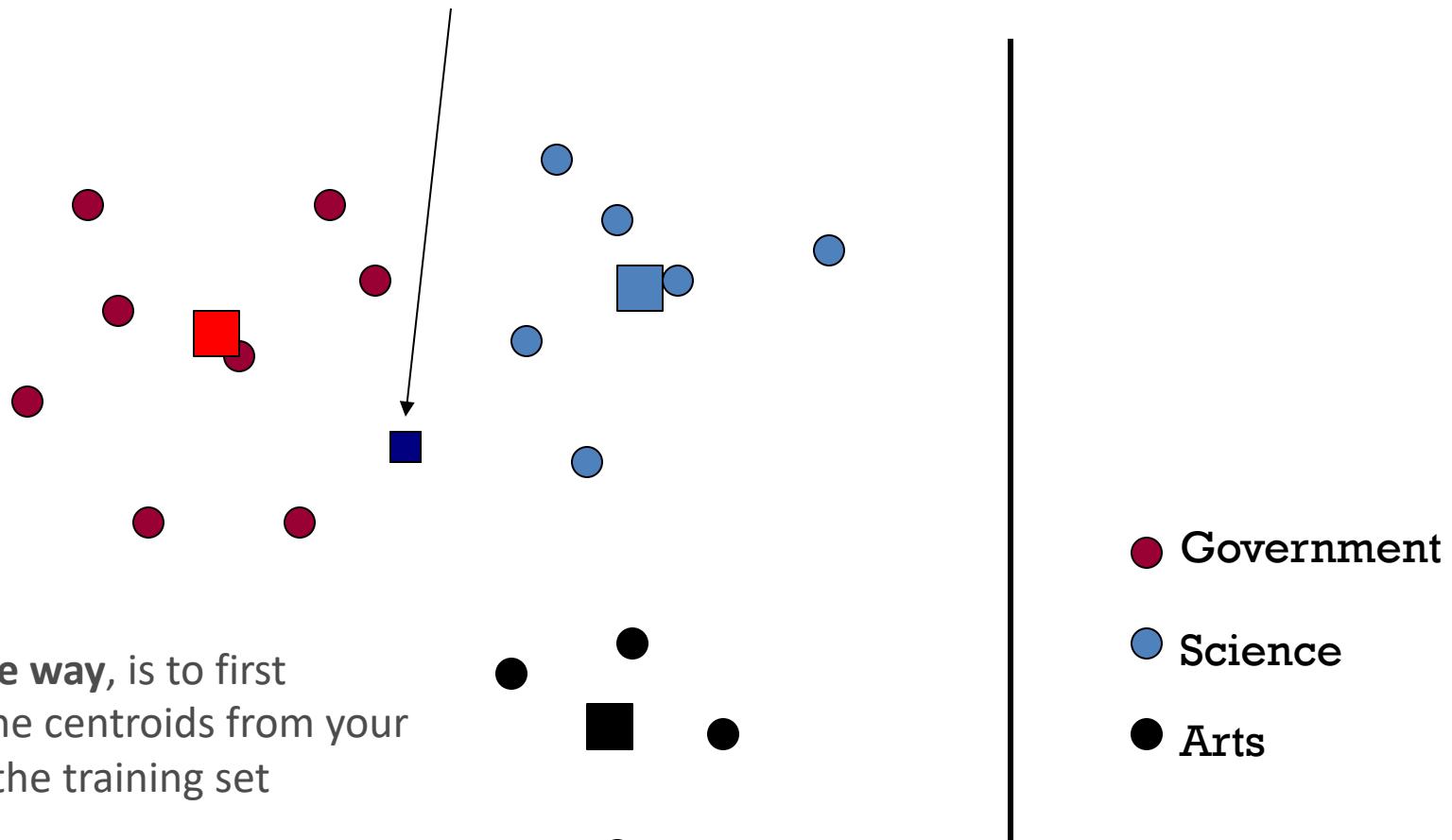
What is the class of this document?



But how can you define these boundaries?

Documents in a Vector Space : simple classifier

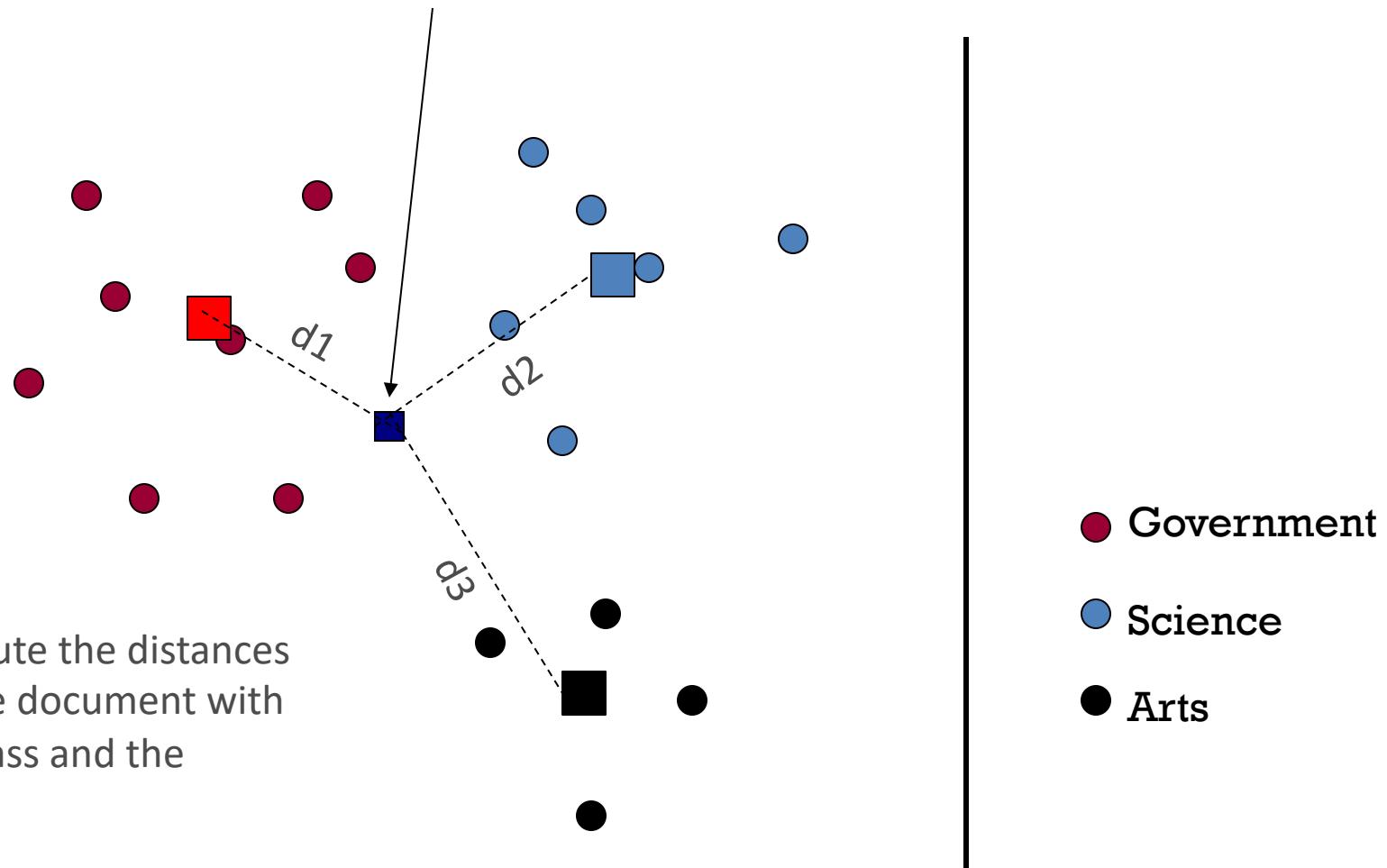
What is the class of this document?



$$\vec{u}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$$

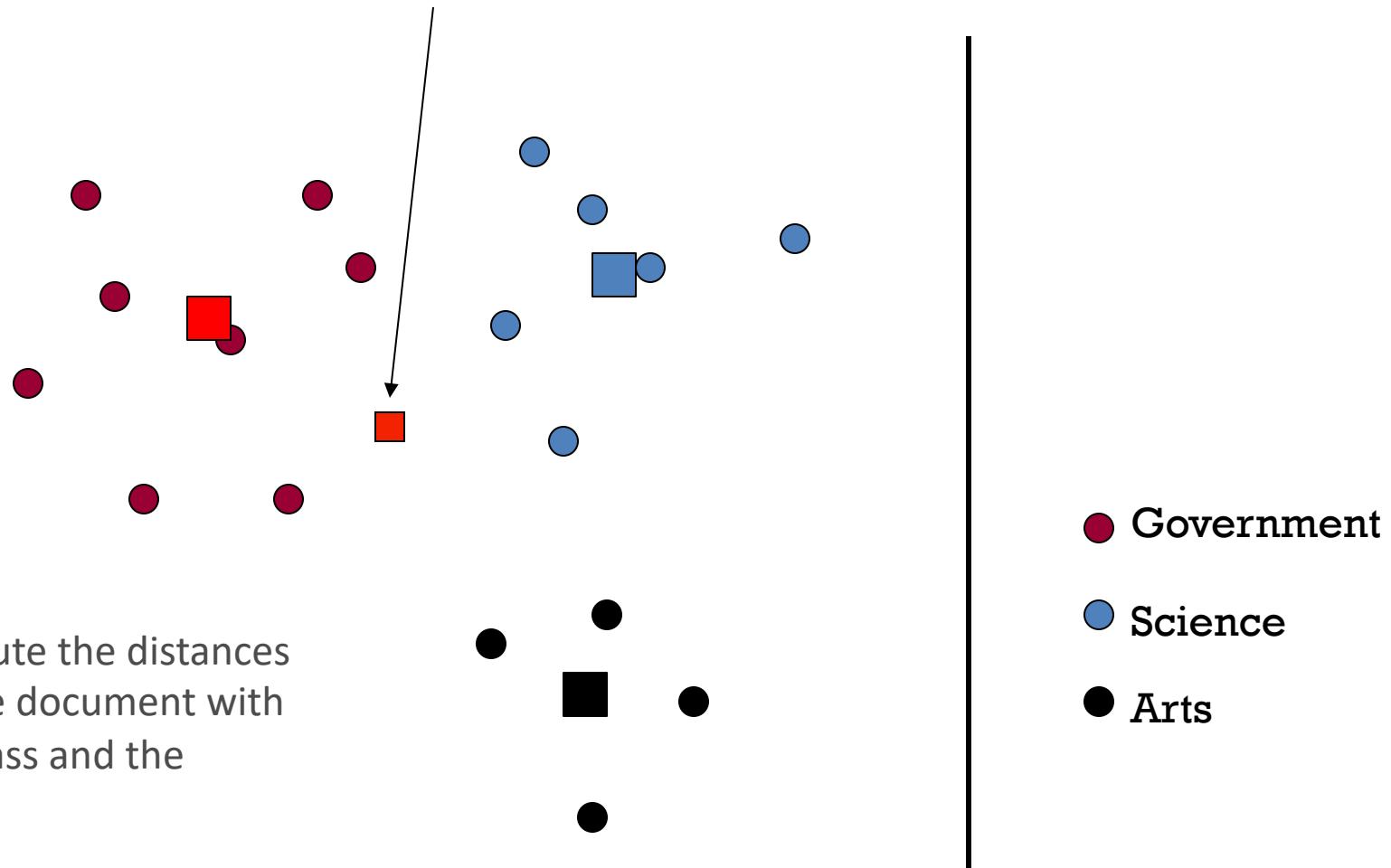
Documents in a Vector Space : simple classifier

What is the class of this document?



Documents in a Vector Space : simple classifier

What is the class of this document? : Government



Then, compute the distances between the document with unknown class and the centroids.

Finally, assign the class as the same class as the nearest centroid.

The Information Retrieval course

Problem Space of this course

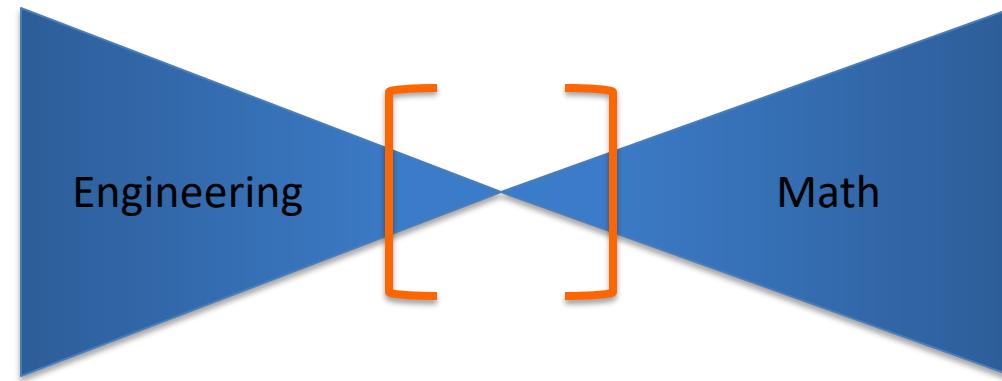
- “Big Data”
- How to
 - collect it
 - index it
 - search it for relevant information

Industry segment of this course

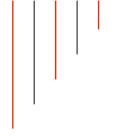


- Search engines
 - Google
 - MS Bing
 - Duck Duck Go
 - nameless others
- Web and social information retrieval = big \$\$\$

Technical content of this course



Goals



- For you to know the fundamentals of information retrieval
- For you to have at your disposal some techniques to find information spread among 10^n documents
- For you to practice Web crawling and **for to build your own simple search engine from scratch**

This course answers three primary questions

- What is information retrieval?
- How to build some of the most important blocks of information retrieval systems?
- How to perform information retrieval in the web?

All together

- Search engines history
- Search & advertising on the Web
- Web corpus
- Characteristics of the Web
- Characteristics of Web search Users
- Search Engine Optimization
- Web crawling
- Index construction
- Map Reduce
- Boolean retrieval
- Scored retrieval
- TF-IDF and corpus-wide statistics
- Text processing
- Link analysis (PageRank, HITS)
- Search Engine Evaluation: Precision, Recall, MAP and NDCG@R

All together

- Search engines history
 - Search & advertising on the Web
 - Web corpus
 - Characteristics of the Web
 - Characteristics of Web search Users
 - Search Engine Optimization
 - Web crawling
 - Index construction
 - Map Reduce
 - Boolean retrieval
 - Scored retrieval
 - TF-IDF and corpus-wide statistics
 - Text processing
 - Link analysis (PageRank, HITS)
 - Search Engine Evaluation: Precision, Recall, MAP and NDCG@R
- You built a complete search engine from the ground up...
- Few professionals had this experience!

Big Data jobs



- Plenty...
- Not just traditional search
 - *making sense of data*
- Google it!
- **Open your own company**
 - Several fields for niche search and recommendation systems
 - Image and video search : nobody is doing it right yet...
 - Deep web search : nobody is doing it right yet...

Where to go from here

- Graphs
- Data mining
- Statistical and Machine learning
- Invest time to learn the fundaments!
 - *Take as many proof-based algorithms courses and proof-based math courses as you can (but avoid taking them at the same time...);*
 - *Always try to do things from scratch (e.g. your own database/webserver/search-engine/etc.);*
 - *Profit from your University years to build your “superpowers”!*

*in the summer vacations, watch Karate Kid (the original)
and pay attention to “wax on wax off...”*

Where to go from here

- Graphs
- Data mining
- Statistical and Machine learning
- Invest time to learn the fundamentals!
 - *Take as many proof-based algorithms courses and proof-based math courses as you can (but avoid taking them at the same time...);*
 - *Always try to do things from scratch (e.g. your own database/webserver/search-engine/etc.);*
 - *Profit from your University years to build your “superpowers”!*

*in the summer vacations, watch Karate Kid (the original)
and pay attention to “wax on wax off...”*

Whenever you need any help, now and forever, just drop by my office (DBH 5058)!

Thank you for your patience!

and see you around!

Information Retrieval