CS186 Discussion #10

(Text Search & Ranking)

Boolean Text Search

- How do you find documents that contain some given words?

- Find all docs matching a Boolean expression:

    - “Database” AND (“Relational” OR “NoSQL”)

Bag of Words Model

- Each document is a multiset of words
  - No stop words (“the”, “to”, “is”, “a”, “<div>”)
  - Stemmed
    - Convert “coded”, “coding”, “coder” to “code”

Bag of Words Model

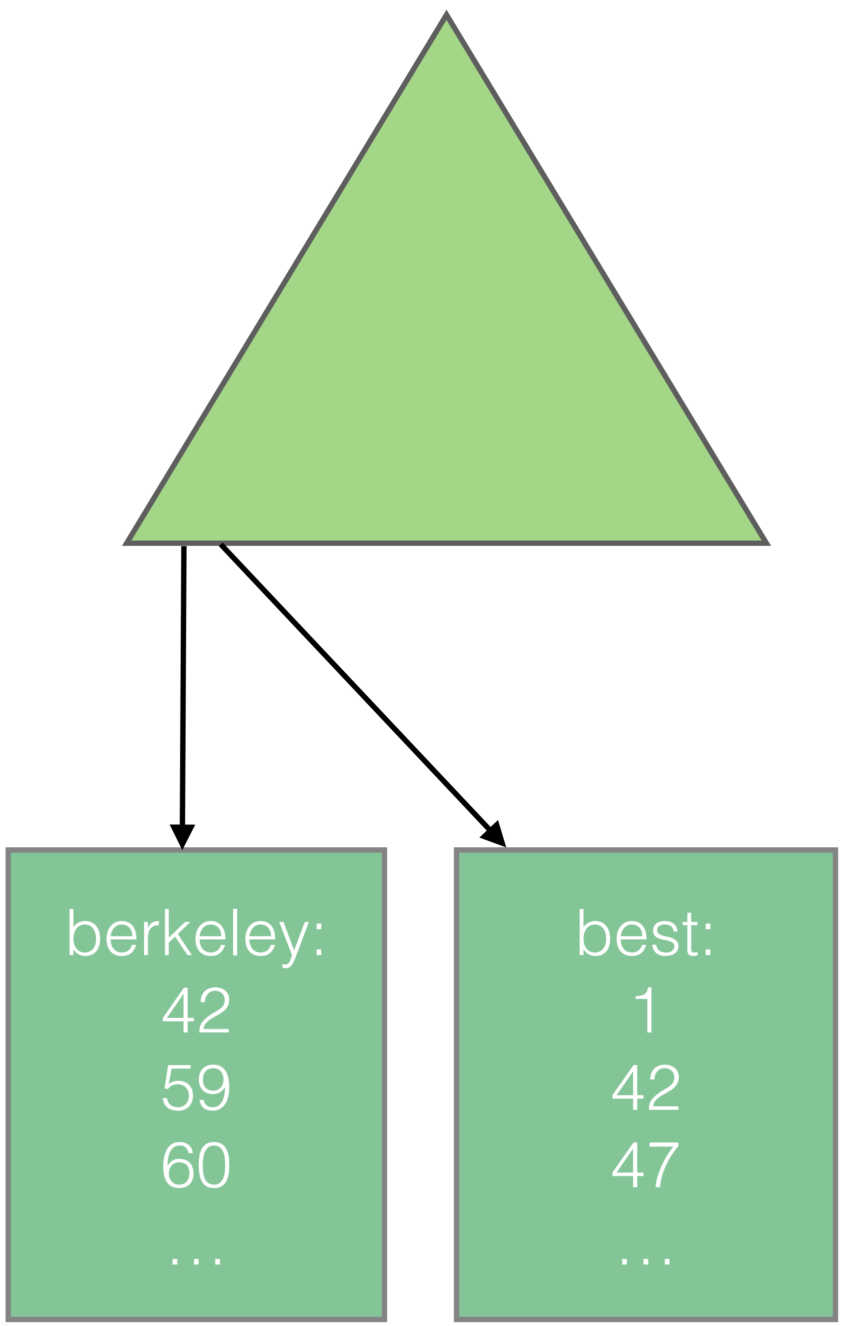
“The quick brown fox jumps over the lazy dog.”

Bag of Words Model

“~~ The~~ quick brown fox jump~~s~~ ~~over~~ ~~the~~ lazy dog~~. ~~”

Inverted Files

- Given a corpus of text files:
  `Files(doc_id string, context string)`
- Create:
  `InvertedFile(term string, doc_id string)`
- Build B+ tree or hash index on `InvertedFile` with `term` as search key



Boolean Search in SQL

```
SELECT IB.docID
   FROM InvertedFile IB, InvertedFile ID,
   InvertedFile IR
   WHERE IB.docID = ID.docID AND
   ID.docID = IR.docID
   AND IB.term = "Berkeley"
   AND ID.term = "Database"
   AND IR.term = "Research"
ORDER BY magic_rank()
```

Searching Phrases

- Example: Looking for “John Smith”
  - Don’t want a document about “John Doe” and “Bob Smith”
- Store position: `InvertedFile(term string, doc_id string, position int)`
- Keep results that are off by 1 position

Boolean Search in SQL

```
SELECT IB.docID
   FROM InvertedFile IB, InvertedFile ID
   WHERE IB.docID = ID.docID
   AND IB.term = “John”
   AND ID.term = “Smith”
   AND (ABS(IB.position - ID.position)=1)
ORDER BY magic_rank()
```

Worksheet Page #1

Which of the following terms would appear in our bag of words if we are "stemming and stopping"?

of

an

<head>

&

lugubrious

nincompoop

headers

Which of the following terms would appear in our bag of words if we are "stemming and stopping"?

~~<head>~~

~~of~~

~~an~~

~~&~~

lugubrious

~~headers~~

nincompoop

Which of following are true about query plans for a conjunctive boolean text search?

- A. It can only be expressed using inner joins.
- B. It must be a left deep query plan.
- C. It can be expressed using only unions.
- D. It must be either right-deep or left-deep.

Which of following are true about query plans for a conjunctive boolean text search?

- A. It can only be expressed using inner joins.
- B. It must be a left deep query plan.
- C. It can be expressed using only unions.
- D. It must be either right-deep or left-deep.

`TermInfo(term string, numDocs int)`

`InvertedFile(term string, docID int, DocTermRank float)`

Assume that ints and floats both require 4 bytes, and strings require 20 bytes. Assume that there are 100,000 documents in the collection, 250,000 unique words, and that the average word appears in 100 documents.

- How big would `TermInfo` be, in bytes?

`TermInfo(term string, numDocs int)`
`InvertedFile(term string, docID int, DocTermRank float)`

Assume that ints and floats both require 4 bytes, and strings require 20 bytes. Assume that there are 100,000 documents in the collection, 250,000 unique words, and that the average word appears in 100 documents.

- How big would `TermInfo` be, in bytes?
  - 250,000 unique words * (20 bytes for term + 4 bytes for numDocs) = 6,000,000 bytes

```
TermInfo(term string, numDocs int)
InvertedFile(term string, docID int, DocTermRank float)
```

Assume that ints and floats both require 4 bytes, and strings require 20 bytes. Assume that there are 100,000 documents in the collection, 250,000 unique words, and that the average word appears in 100 documents.

- How big would `InvertedFile` be, in bytes?

`TermInfo(term string, numDocs int)`

`InvertedFile(term string, docID int, DocTermRank float)`

Assume that ints and floats both require 4 bytes, and strings require 20 bytes. Assume that there are 100,000 documents in the collection, 250,000 unique words, and that the average word appears in 100 documents.

- How big would `InvertedFile` be, in bytes?
  - Number of rows: 250,000 words * 100 docs per word = 25e6
  - Number of bytes per row: 20 + 4 + 4 = 28
  - Total bytes : 25e6*28 = 7e8 bytes

Ranking

- How to order output from text search
- What makes search engines different
- Combination of statistics, linguistics, graph theory

Vector Space Model

- Each document and query is a vector
  - For 100,000 words in dictionary, use 100,000 dimensional vector/array
  - Each index represents one word, and is the count for that word
- Similarity between two documents is distance between vectors

Vector Space Model

Dictionary

quick
fox
jump
dog
database

Context

The quick fox jumped over the quick dog.

Bag of Words

quick, fox, jump, quick, dog

Vector Space Model

Dictionary

quick
fox
jump
dog
database

Context

The quick fox jumped over the quick dog.

Bag of Words

quick, fox, jump, quick, dog

<2, 1, 1, 1, 0>

Vector Space Model

Dictionary

quick
fox
jump
dog
database

Query

“quick fox”

<1, 1, 0, 0, 0>

Doc 1

“quick fox fox quick quick fox”

<3, 3, 0, 0, 0>

Doc 2

“database”

<0, 0, 0, 0, 1>

Vector Space Model

Dictionary

quick
fox
jump
dog
database

Query

"quick fox"

<1, 1, 0, 0, 0>

Doc 1

"quick fox fox quick quick fox"

<3, 3, 0, 0, 0>

Doc 2

"database"

<0, 0, 0, 0, 1>

Distance between query and doc1: sqrt(8)
Distance between query and doc2: sqrt(3)

Cosine Similarity

- Normalize each dimension/index by vector's length
- Finding Euclidean distance of normalized vector is same as finding $A \cdot B$
  - $A \cdot B = ||A||\ ||B|| \cos \theta$
- Angle between two similar documents small if many tokens in common, because vectors pointing in same direction
  - small θ -> larger $\cos \theta$, larger similarity ranking

TF-IDF

- Want to favor repeated terms in the document
- Want to favor unusual/uncommon words
  - “ISBN-13: 978-0072465631”
- Term frequency (TF): occurrences of term t in document d
- Inverse doc frequency (IDF):

  log(total # docs/# of docs with term t)
- DocTermRank: TF * IDF

TF-IDF

Dictionary

quick
fox
jump
dog
database

Doc 1

"quick fox fox
quick quick fox"

<3, 3, 0, 0, 0>

Doc 2

"database"

<0, 0, 0, 0, 1>

TF and IDF of "quick" for doc 1?

TF-IDF

Dictionary

quick
fox
jump
dog
database

Doc 1

“quick fox fox quick quick fox”

<3, 3, 0, 0, 0>

Doc 2

“database”

<0, 0, 0, 0, 1>

TF and IDF of “quick” for doc 1?

TF: 3

IDF: log(2/1)

Worksheet Pages #2, 3

Term	Term Count
this	1
is	1
a	2
sample	1

Term	Term Count
this	1
is	1
another	2
example	3

- What is the TF-IDF of “this” in document 1?

Term	Term Count
this	1
is	1
a	2
sample	1

Term	Term Count
this	1
is	1
another	2
example	3

- What is the TF-IDF of “this” in document 1?
- TF(“this”, doc1) = 1
- IDF(“this”, doc1) = log(2/2) = 0
- TF-IDF(“this”, doc1) = 1*0 = 0

Term	Term Count
this	1
is	1
a	2
sample	1

Term	Term Count
this	1
is	1
another	2
example	3

- What is the TF-IDF of “example” in document 2?

Term	Term Count
this	1
is	1
a	2
sample	1

Term	Term Count
this	1
is	1
another	2
example	3

- What is the TF-IDF of “example” in document 2?
- TF(“example”, doc2) = 3
- IDF(“example”, doc2) = log(2/1) = 0.30
- TF-IDF(“example”, doc2) = 3*0.30 = 0.90

In general, what is characteristic of (term, doc) which have TF-IDF = 0?

In general, what is characteristic of (term, doc) which have TF-IDF = 0?

- They appear in all documents, so IDF = 0, and thus TF-IDF = 0.

We want to join with the Docs table to return the actual contents of the pages.
Which of the following join techniques is best for this join?

- A. Sort Merge Join
- B. Nested Loops Join
- C. Hash Join
- D. Index Nested Loops Join

We want to join with the Docs table to return the actual contents of the pages.
Which of the following join techniques is best for this join?

- A. Sort Merge Join
- B. Nested Loops Join
- C. Hash Join
- D. Index Nested Loops Join

We want to join with the Docs table to return the actual contents of the pages.
Which of the following join techniques is best for this join?

- D. Index Nested Loops Join
- INLJ is streaming, so we aren’t reading required to read the entire Docs file
- INLJ is lazy so we can return results as they are requested (good for paginated web search)
- INLJ is easy to parallelize: issue multiple requests for documents at once (given the DocIDs)

Given the following documents: Document 1: <6, 8, 0, 0>, Document 2: <0, 0, 24, 10>, Document 3: <0, 0, 3, 4>

Why is it important to normalize our document vectors?

Given the following documents: Document 1: <6, 8, 0, 0>, Document 2: <0, 0, 24, 10>, Document 3: <0, 0, 3, 4>
Why is it important to normalize our document vectors?

- Ranking becomes biased by document length without normalization -- long documents are more similar to each other than short documents.

- Plus, cosine similarity is easier to compute.

Given the following documents: Document 1: <6, 8, 0, 0>, Document 2: <0, 0, 24, 10>, Document 3: <0, 0, 3, 4>
We send the query described by the vector <1, 0, 2, 2> to our search engine. In what order will these documents be ranked?

Given the following documents: Document 1: <6, 8, 0, 0>, Document 2: <0, 0, 24, 10>, Document 3: <0, 0, 3, 4>
We send the query described by the vector <1, 0, 2, 2> to our search engine. In what order will these documents be ranked?

- Normalize:
  - Document 1: <⅗, ⅘, 0, 0>
  - Document 2: <0, 0, 12/13, 5/13>
  - Document 3: <0, 0, ⅗, ⅘>
  - Query: <⅓, 0, ⅔, ⅔>

- After computing dot products, ranking is Doc 3, 2, 1.

What are the pros and cons of partitioning our inverted files by term?

What are the pros and cons of partitioning our inverted files by term?

- Pros: Not all machines must work on every query -- only need results from a few machines.
- Cons: Load balancing:
  - (1) terms are nonuniformly distributed, so may be difficult to partition them up evenly per machine
  - (2) a query will only have a few terms associated with it, which means only a few machines work at a time

What are the pros and cons of partitioning our inverted files by document?

What are the pros and cons of partitioning our inverted files by document?

- Pros: Easy to uniformly distribute all documents and load balance.
- Cons: All machines must work on every query -- must wait for results from every machine before answering query.