IR1: Introduction to Information Retrieval
Mid-Semester: Where are we?

- So far, we have talked about fundamentals of Web Systems
  - Protocols for communication (like HTTP, TCP)
  - Building blocks (request/response cycle, JSON objects, RESTful paradigm)
  - Web site behavior (GET/POST requests, servers, clients)

- You’ve done full stack development (P1, P2, P3)
  - Flask servers, HTML, CSS, JavaScript
  - React framework, node, pip
  - Virtual environments, bash scripting, selenium

- We’ve talked about distributed systems
  - MapReduce, GFS, Blockchain: applications of web systems beyond websites
Review: Blockchain

- **Blockchain** is a type of distributed system that enables **transactions** to occur within a **distributed ledger**
  - Blockchain allows for multiple participants in the system to gain a reliable view of the transaction ledger
  - Blockchain serves as the basis for enabling **cryptocurrencies** like Bitcoin and Litecoin
- **Cryptocurrency**: report transactions between **wallets**
  - Implementation: public key infrastructure. Each sender/recipient signs transactions
- **Idea**: **incentivize** participants in the network to correctly log transactions
  - (Loosely) transactions broadcasted everywhere
    - Miners try hashing batches of transactions
    - Eventually, one miner finds a hash, tells everyone else, providing a **proof of work**
    - That **block** gets added to the chain!
One-Slide Summary: Information Retrieval I

• The problem of **information retrieval** is construed as:
  • Given a **query document** and a **corpus of documents**, find all **documents** that relate to the query
    • Want *all* documents that relate to the query (no “false negatives”)
    • Want *only* the documents that relate to the query (no “false positives”)

• IR systems **rank order** documents from a corpus by relatedness
  • We pick the *top k* documents according to some algorithm
    • Spoiler: PageRank by Google might be a solid choice

• We **represent** documents as vectors of numbers to help develop **similarity** metrics
  • How related are these two documents?
  • Then, you rank them by similarity score
Google Search

• Somewhat profitable company you may have heard of
• Searching for things you want is **hard**
• Consider: if you search for “Dijkstra’s algorithm”
  • If you search through *all documents*
    • Do you care about Dijkstra’s personal history or biography?
    • Do you care about other algorithms?
    • Dijkstra’s algorithm is graph-based... maybe you want other graph algorithms?
Document Search

• If you were Google, what would you need to do?
  • Search bar lets users type in a **query**
  • But what do you use as the document **corpus**?
    • Basically download the internet (good luck lol)
      • Can you do this dynamically?
    • For a future lecture: apply search engine optimization (SEO) to influence how your site gets represented in a search engine’s corpus

• Given a **query**, iterate through all documents in the **corpus** (or **index**), compute **similarity** between query and document
  • Count occurrences of query tokens in each document?
  • Count documents where query appears one or more times?
  • Count documents where query *doesn’t* appear?
Key problem: ranking results

• 33% clicks on top result
  • “I’m feeling lucky”
  • It’s not enough to get the best document as the 100th result

• Different ranking methods
  • Words on page

• Importance of page using links
Goal of ranking algorithms

• Which web pages (documents) does the person searching want to find?
  • Given query “kangaroos”? “animals”?

Kangaroos live in Australia and jump.

Cows live all over the world. Unlike kangaroos, they cannot jump.

Aluminum foil is shiny.
Thought questions

• What is the role of the corpus/index? Why would searching be difficult without one?

• Why might the structure of the corpus/index change depending on which ranking algorithm we use?
Boolean Retrieval

• Does the **query** appear in a **document**?
  • Boolean decision. No counting, no sorting
  • “Kangaroos”

• Augment: composition of tokens in query
  • “Kangaroos AND NOT cows”

• Basically: evaluate a Boolean predicate for each document.
  • If true: document returned
  • If false: document ignored
Boolean Retrieval

• Query: Kangaroos

Kangaroos live in Australia and jump.

Cows live all over the world. Unlike kangaroos, they cannot jump.

Aluminum foil is shiny.

1 1 0
Boolean Retrieval

• Query: Kangaroos and NOT cows

Kangaroos live in Australia and jump.

Cows live all over the world. Unlike kangaroos, they cannot jump.

Aluminum foil is shiny.

1 0 0
Index Construction for Boolean retrieval

- **Inverted index**: words to documents

<table>
<thead>
<tr>
<th>Document 0</th>
<th>Document 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kangaroos can jump.</td>
<td>Cows can not jump.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>kangaroo</td>
<td>0</td>
</tr>
<tr>
<td>can</td>
<td>0, 1</td>
</tr>
<tr>
<td>jump</td>
<td>0, 1</td>
</tr>
<tr>
<td>cow</td>
<td>1</td>
</tr>
<tr>
<td>now</td>
<td>1</td>
</tr>
</tbody>
</table>
Boolean Retrieval using Inverted Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>kangaroo</td>
<td>0</td>
</tr>
<tr>
<td>can</td>
<td>0, 1</td>
</tr>
<tr>
<td>jump</td>
<td>0, 1</td>
</tr>
<tr>
<td>cow</td>
<td>1</td>
</tr>
<tr>
<td>now</td>
<td>1</td>
</tr>
</tbody>
</table>

• Given **kangaroo AND NOT cow**
  • “Kangaroo” -> document 0
  • “NOT cow” -> document 0
  • AND operation
    • Document 0

• What are the benefits/drawbacks of Boolean retrieval?

• What if you have a giant index?
  • Related: from earlier, why is an index necessary here?
Vector Space Model

• Boolean retrieval is not particularly rich
  • Yes/no per document, but not really a way to rank results

• We can embed documents into a vector space to allow relating two documents by measuring properties of the vector
  • Notably, what is the angle formed by the two vectors
Boolean index to vectors

<table>
<thead>
<tr>
<th>Term</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>kangaroo</td>
<td>0</td>
</tr>
<tr>
<td>can</td>
<td>0, 1</td>
</tr>
<tr>
<td>jump</td>
<td>0, 1</td>
</tr>
<tr>
<td>cow</td>
<td>1</td>
</tr>
<tr>
<td>now</td>
<td>1</td>
</tr>
</tbody>
</table>

- Assign each term to be indices in a vector
  - For each document \( i \), we assign \( i_j = 1 \) if document \( i \) contains term \( j \)

- Btw: what about ordering?

- Document 0: \([1, 1, 1, 0, 0]\)
- Document 1: \([0, 1, 1, 1, 1]\)

- You can work backwards:
  \([1, 0, 1, 0, 1] = ?\)
Documents as vectors

• A document is a vector

• Each dimension represents a word
  • Doesn’t capture order
  • Doesn’t capture counts

• # of dimensions: # of unique words in all documents
  • How many words are there?
  • Also, for a future lecture: should “kangaroo” and “kangaroos” be treated the same?
Document Similarity in Vector Space

• Want: Number that captures how “close” two vectors are

• *Vector space similarity* can be represented with a *cosine* of the angle between two vectors

• Recall that cosine:
  • Depends on two adjacent vector lengths
  • =1 when angle is zero (points are identical)
  • *Smaller* when angle is *greater*
  • *Larger* when angle is *lesser*
Vector space similarity (and demo)

• Euclidean dot product formula for computing cosine similarity

\[
\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \|\mathbf{B}\| \cos \theta
\]

\[
similarity = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}
\]

\[
Sim(D_i, D_j) = \sum_{k=1}^{t} w_{ik} * w_{jk}
\]

• (protip: import numpy and import math)
Thought Questions

• If the query is "cows are cool", is the word "cool" represented in the vector made from the query? Does that matter?

• What is the index data structure if we're using vectors to represent documents?

• In general, will there be more 0s or 1s in most vectors?
Thought Questions

• If the query is "cows are cool", is the word "cool" represented in the vector made from the query? Does that matter?
  • If a query contains a token not represented in the index, you ignore it
  • Whether it matters depends or not. If a user searches for “cows are cool,” perhaps they don’t want an article about “cows are awful”

• What is the index data structure if we're using vectors to represent documents?

• In general, will there be more 0s or 1s in most vectors?
  • 1’s are pretty sparse (consider: how many words are there?)
Next: Adding More Information to Vectors

• Right now, vectors contain only 0 or 1 depending on whether they contain a word

\[ \text{doc1} = [1, 1, 1, 0, 0] \]

• What else can be used to enhance the information conveyed by the vector?
  • Recall our goal: compare query to document for ranking similarity

• Cunning plan: let’s try using real numbers instead of 0/1
Term frequency

- We can count the number of times a term appears in a document
- A term frequency is the number of times a word appears in each document!

Kangaroos can jump. Kangaroos live in Australia.

Unlike kangaroos, cows cannot jump. Cows like eating grass.

<table>
<thead>
<tr>
<th>Term</th>
<th>Term Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kangaroos</td>
<td>2</td>
</tr>
<tr>
<td>Can</td>
<td>1</td>
</tr>
<tr>
<td>Jump</td>
<td>1</td>
</tr>
<tr>
<td>Live</td>
<td>1</td>
</tr>
<tr>
<td>In</td>
<td>1</td>
</tr>
<tr>
<td>Australia</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Term Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlike</td>
<td>1</td>
</tr>
<tr>
<td>Kangaroos</td>
<td>2</td>
</tr>
<tr>
<td>Cows</td>
<td>2</td>
</tr>
<tr>
<td>Cannot</td>
<td>1</td>
</tr>
<tr>
<td>Jump</td>
<td>1</td>
</tr>
<tr>
<td>Like</td>
<td>1</td>
</tr>
<tr>
<td>Eating</td>
<td>1</td>
</tr>
<tr>
<td>Grass</td>
<td>1</td>
</tr>
</tbody>
</table>
Problems with Term Frequency

- Across large corpora, some words appear more frequently than others.
- Consider transcriptions of Kevin’s lectures. How many times does “uh” appear?
Problems with Term Frequency

• Since “the” occurs so frequently, it will bias vectors
  • Two unrelated vectors may have a large component due to “the”

• Below: TFs of common words in “fake news” tweets. Note “the”, “to”, and “of” dominate all the vectors, even if each tweet is unrelated

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 |
• A word’s **Document Frequency** is the fraction of documents in which that word appears
  • We can characterize a word’s rarity by its DF (lower = rarer)
  • Usually, we use **inverse document frequency** (IDF = 1/DF) for rarity

<table>
<thead>
<tr>
<th>Term</th>
<th>DF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kangaroos</td>
<td>1/3</td>
</tr>
<tr>
<td>Australia</td>
<td>3/3</td>
</tr>
<tr>
<td>In</td>
<td>2/3</td>
</tr>
</tbody>
</table>

Kangaroos can jump. Kangaroos live in Australia.

Australia is in the southern hemisphere.

One of the longest flights is from London to Sydney, Australia.
Problems with Inverse Document Frequency

• Suppose we return documents that contain terms from the query with high IDF
  • e.g., if my query is "Rumpelstiltskin", odds are very few documents contain that word (high IDF), so just return those documents

• Problem: how do I scale IDFs among search terms?
• Consider the query: "Rumpelstiltskin book"
  • What if "Rumpelstiltskin" appears 10 out of 29 million documents and "book" appears in 1 million out of 29 million?
  • Is the term "book" 100,000 times less useful than "Rumpelstiltskin"?
• Solution: apply log. \( IDF = \log\left(\frac{N}{n_k}\right) \)
  (\( n_k \) is # documents containing \( k \))
  (\( N \) is total number of documents)
Term Frequency - Inverse Document Frequency

• Idea: Combine TF and IDF for best of both worlds:
  • TF: pull documents that contain lots of instances of query tokens
  • IDF: pull documents according to rarity of given query tokens
  • “tf-idf”

\[ w_{ik} = tf_{ik} \times \log \frac{N}{n_k} \]

• \( T_k \) = term \( k \) in document \( i \) \( (D_i) \)
• \( tf_{ik} \) = term frequency of \( T_k \) in \( D_i \)
• \( N \) = number of documents in corpus
• \( n_k \) = number of documents containing \( T_k \)
tf-idf applies to each term in each document

Documents (i) 0 and 1

- Kangaroos can jump. \( i=0 \)
- Kangaroos live in Australia. \( i=1 \)

Words:

\[
\begin{bmatrix}
\text{kangaroos} & \text{can} & \text{jump} & \text{live} & \text{in} & \text{australia}
\end{bmatrix}
\]

\( k=0 \quad k=1 \quad k=2 \quad k=3 \quad k=4 \quad k=5 \)

tf-idf vectors

\[
D_0 = [w_{00}, w_{01}, w_{02}, w_{03}, w_{04}, w_{05}]
\]

\[
D_1 = [w_{10}, w_{11}, w_{12}, w_{13}, w_{14}, w_{15}]
\]
Computing td-idf over a corpus

• IDF is independent of a specific document, and applies to each word
  • When processing a corpus, you can compute IDF’s for each word in vocabulary
  • Then, just multiply by tf when considering each document

• IDF basically becomes a coefficient for each word $T_k$ in the vocabulary

<table>
<thead>
<tr>
<th></th>
<th>$T_0$</th>
<th>$T_1$</th>
<th>...</th>
<th>$T_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>$tf_{00}IDF_0$</td>
<td>$tf_{01}IDF_1$</td>
<td>...</td>
<td>$tf_{0k}IDF_k$</td>
</tr>
<tr>
<td>D1</td>
<td>$tf_{10}IDF_0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dn</td>
<td>$tf_{n0}IDF_0$</td>
<td></td>
<td></td>
<td>$tf_{nk}IDF_k$</td>
</tr>
</tbody>
</table>

| IDF0 | IDF1 | IDF2 | IDF3 |
tf-idf normalization

• If you have a longer document, \textbf{tf} terms are naturally higher than in shorter documents (why?)

• Normalize term weights
  • Longer documents not given more weight
  • Normalize to sum-of-squares

\[
    w_{ik} = \frac{t_f \cdot \log(N/n_k)}{\sqrt{\sum_{k=1}^{t} (t_f \cdot \log(N/n_k))^2}}
\]

• Some references use non-normalized tf-idf
  • \( w_{ik} = t_f \cdot \log(N/n_k) \)
Vector space similarity

• Similarity of two docs is:

\[ \text{Sim}(D_i, D_j) = \sum_{k=1}^{t} w_{ik} \cdot w_{jk} \]

\[ \text{sim}(d_j, q) = \frac{d_j \cdot q}{\|d_j\| \|q\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}} \]

Normalized ahead of time, when computing term weights.

Not normalized ahead of time
Thought questions (compared to binary retrieval)

• What is one reason why term frequency gives us more information than only a 1 or 0?

• What is one reason why inverse document frequency gives more information than a 1 or 0?

• If you knew that a search engine used tf-idf, what is a strategy that could help your site become a top result?