CSCE 670 - Information Storage and Retrieval

Lecture 2: Boolean Retrieval

Yu Zhang

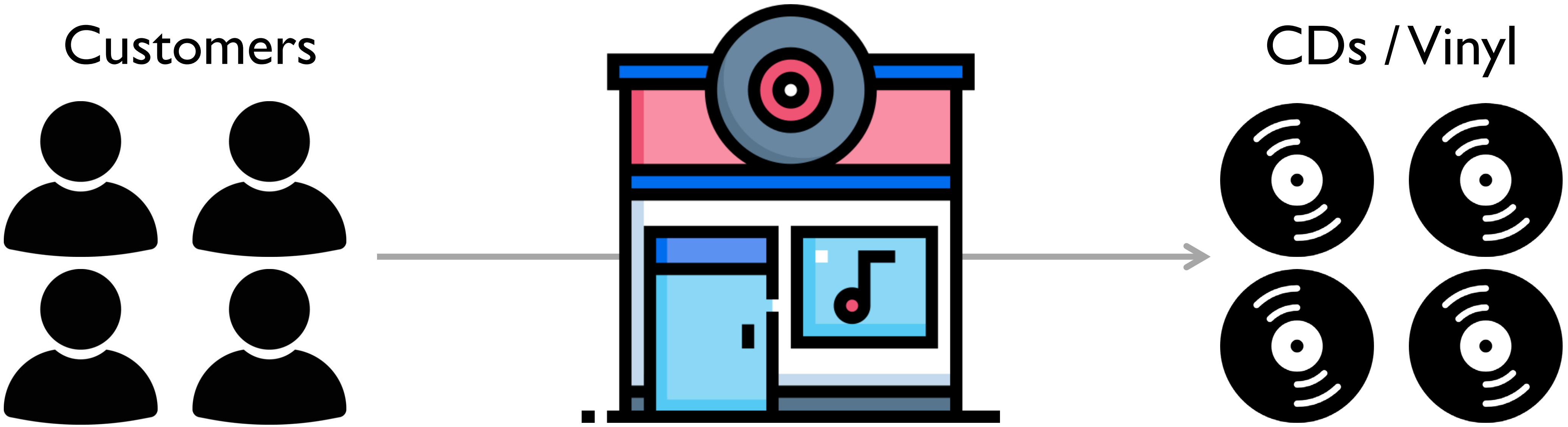
yuzhang@tamu.edu

August 28, 2025

Course Website: https://yuzhang-teaching.github.io/CSCE670-F25.html

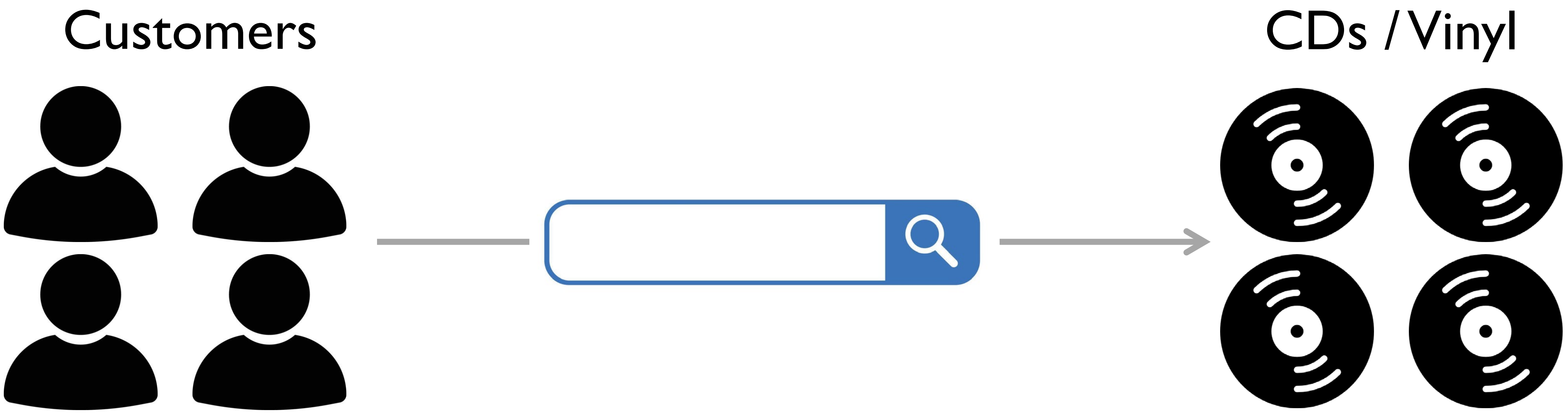
Adapted from the slides by Prof. James Caverlee

We are opening a record store!



- Over the course of the semester, we will progressively build up the search and recommendation capabilities of our store.
  - This + next few weeks: Focus on search basics
  - Then: Focus on recommendation basics
  - Later: Revisit both search and recommendation via advanced topics (e.g., LLMs)

This + Next Few Weeks: Help Users Search our Store



Basic Concepts

- Information need
  - *"I want Taylor Swift's latest album."*
- Query
  - *"Taylor Swift's latest album"*
  - *"Taylor Swift album 2025"*
  - …
- Documents
  - A pool of candidates (e.g., CDs)
  - Some candidates may satisfy the information need.
  - Each candidate is associated with some text information.
    - *"Artist: Taylor Swift; Lyrics: Meet me at midnight, Staring at the ceiling with you …"*

Basic Concepts

- Query Representation
  - If we want to design an automated algorithm for search, we need a way to represent the query that a computer can understand.
  - *[0, 1, 1, 0, …]*
  - *[0.255, -1.342, …]*
- Document Representation
  - We need to use a similar way to represent each "document" (i.e., candidate).
- Relevance Function
  - How can we decide which "document" can satisfy the information need?
  - *Relevance(Query, Document1) = 0.8*
  - *Relevance(Query, Document2) = 0.5*
  - …

Key Challenges Motivating Much of this Course

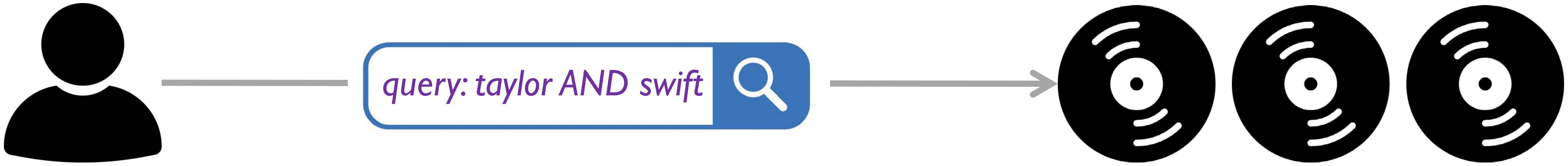
- How do we represent our queries and documents?
  - What is our "representation function"?

- What is our relevance function?

- How do we know if we are doing a good job?

Today: Simplifying Assumptions

- Our store front-page only supports Boolean keyword queries.
  - "*karma*"
  - "*love OR song*"
  - "*taylor AND swift*"
  - "*taylor AND (NOT swift)*"
  - …
- Based on these queries, we return a list of matching albums.



We return a set of matching albums. (No rank order!)

Example Album

- Artist: *Taylor Swift*
- Album Title: *Midnights*
- Year: *2022*
- Track Listing: *Lavender Haze, Maroon, Anti-Hero, …*
- Lyrics: *Meet me at midnight, Staring at the ceiling with you, Oh, you don't ever say too …*

What do our users want to search for?

- Another example where each “document” has multiple fields: Course Explorer

Title	CRN Syllabus	S..	Crse	Sect	Hrs	Instructor(s)	Meeting Times
PROGRAMMING I	45405 Syllabus	CSCE	110	500	4	Ki Hwan K. Yum (P)	Su M T W R F S 12:40 PM-01:30 PM **Type:** Lecture **Building:** ZACH **Room:** 350 **Date:** 08/25/2025 - 12/16/2025 Su M T W R F S 02:35 PM-03:25 PM **Type:** Laboratory **Building:** ZACH **Room:** 596 **Date:** 08/25/2025 - 12/16/2025

- What parts of an album do we index?
- How to index these parts?
  - Everything in one index?
  - Some parts in one index?
  - Each facet in its own separate index?

One More Simplifying Assumption

- Record store front-page only supports Boolean keyword queries over song lyrics.

- Artist: *Taylor Swift*
- Album Title: *Midnights*
- Year: *2022*
- Track Listing: *Lavender Haze, Maroon, Anti-Hero, …*
- Lyrics: *Meet me at midnight, Staring at the ceiling with you, Oh, you don't ever say too …*

A Simple (but not Good) Boolean Retrieval Algorithm

- Query: *midnight AND staring*
- Algorithm Pseudo Code:

```
results = []
For CD in CDs:
    If "midnight" in CD.lyrics and "staring" in CD.lyrics:
        results.append(CD)
return results
```

A Simple (but not Good) Boolean Retrieval Algorithm

- Query: *believe OR (NOT love)*
- Algorithm Pseudo Code:

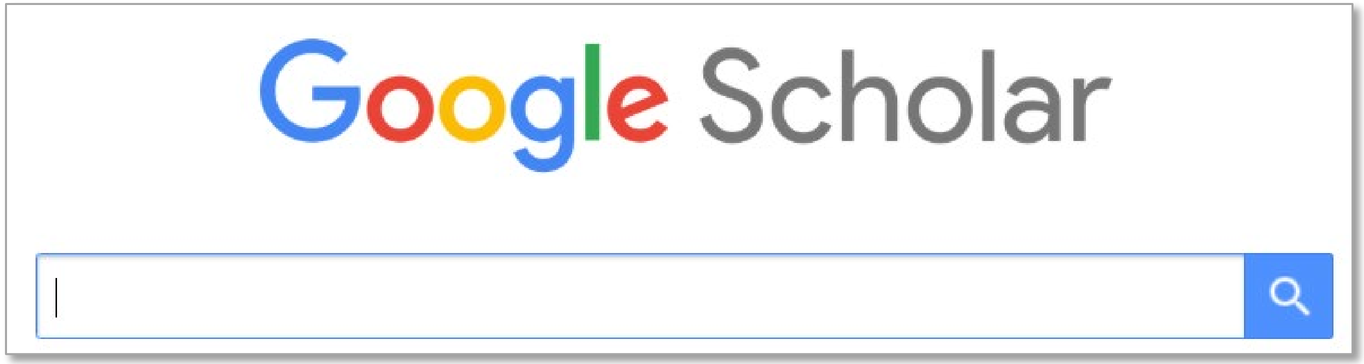
```
results = []
For CD in CDs:
    If "believe" in CD.lyrics or "love" not in CD.lyrics:
        results.append(CD)
return results
```

- What if a new CD arrives in our store?

```
CDs.append(new_CD)
```

Problems?

- For each query, we need to scan all the documents.
  - If there are M documents in total, and each document has N words on average, the time complexity will be $O(MN)$.



- 240,000,000+ papers on the Web by the end of 2019 [1].
- Let's assume that the title and abstract of each paper contain about 200 words.

  - Scanning 48 billion words for each query!
  - If you had to wait several minutes every time to get the paper you are looking for, would you still use this academic search engine?

[1] *Microsoft Academic Graph: When Experts are Not Enough.* Quantitative Science Studies 2020.

Inverted Index: A More Efficient Solution

- Document 1 (d1): "*any choose love*"
- Document 2 (d2): "*zebra any love*"
- …

Vocabulary
any
believe
choose
love
midnight
starring
zebra

Inverted Index: A More Efficient Solution

- Document 1 (d1): "*any choose love*"
- Document 2 (d2): "*zebra any love*"
- …

Vocabulary		
any	→	d1
believe		
choose	→	d1
love	→	d1
midnight		
starring		
zebra		

Inverted Index: A More Efficient Solution

- Document 1 (d1): "*any choose love*"
- Document 2 (d2): "*zebra any love*"
- …

Vocabulary				
any	→	d1	→	d2
believe				
choose	→	d1		
love	→	d1	→	d2
midnight				
starring				
zebra	→	d2		

Inverted Index: A More Efficient Solution

Vocabulary								
any	→	d1	→	d2	→	d5		
believe	→	d4						
choose	→	d1	→	d5	→	d6	→	d10
love	→	d1	→	d2	→	d8		
midnight	→	d7	→	d9				
starring	→	d7	→	d9				
zebra	→	d2	→	d8				

- Query: "*any*"
  - {d1, d2, d5}
- Query: "*any AND zebra*"
  - {d1, d2, d5} ∩ {d2, d8} = {d2}

Inverted Index: A More Efficient Solution

Vocabulary									
any	→	d1	→	d2	→	d5			
believe	→	d4							
choose	→	d1	→	d5	→	d6	→	d10
love	→	d1	→	d2	→	d8			
midnight	→	d7	→	d9					
starring	→	d7	→	d9					
zebra	→	d2	→	d8					

- Query: "*believe OR midnight*"
  - {d4} ∪ {d7, d9} = {d4, d7, d9}
- Query: "*any AND (NOT zebra)*"
  - {d1, d2, d5} – {d2, d8} = {d1, d5}

Questions?

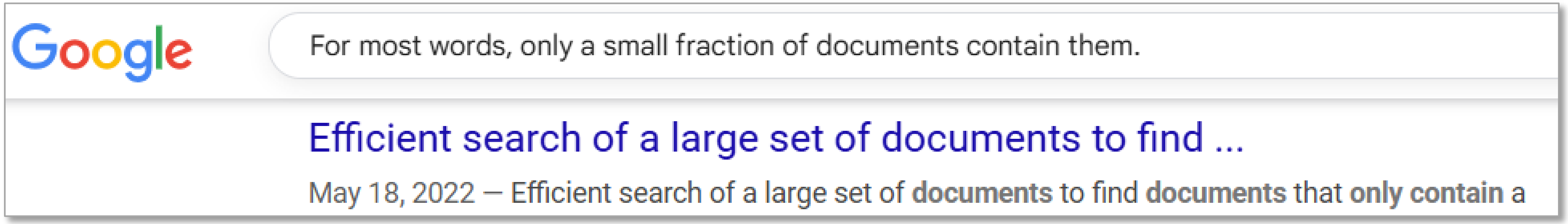
Inverted Index: A More Efficient Solution

Vocabulary								
any	→	d1	→	d2	→	d5		
believe	→	d4						
choose	→	d1	→	d5	→	d6	→	d10
love	→	d1	→	d2	→	d8		
…								

- What if a new CD (d1000) arrives in our store?
  - Scan its lyrics and update our inverted index
  - Move the scanning process into index construction, so it does not take up time during the on-the-fly processing of user queries.
  - Moreover, the data only needs to be scanned once, instead of being scanned for every query.

Inverted Index: A More Efficient Solution

- What is the time complexity of the Boolean retrieval process now?
  - Number of words in the query: usually less than 20
  - $\times$ Time to access the linked list of a word: $O(1)$
  - $+$ Set operations based on these linked lists: proportional to the total length of these linked lists (i.e., the total number of documents containing these words)
    - For most words, only a small fraction of documents contain them.
    - The remaining words (i.e., stop words) appear in many documents and are typically considered uninformative, so they are usually ignored.



Summary: Boolean Retrieval

- Advantages
  - Precise, if you know the right strategies (e.g., how to iteratively refine your queries, use of boolean operators, …)
  - Typically, efficient in practice

- Disadvantages
  - Users must understand Boolean logic!
  - Boolean logic does not capture language richness.
  - Feast or famine in results: often get 0 results or 1000s
  - Result sets are unordered.
  - What about partial matches? E.g., a document does not exactly match the query but it is "close"?

Can we improve the index?

- To support phrase queries?
  - “*taylor swift*”, “*670 homework solution*”
- To support proximity queries?
  - “*taylor NEAR:2 swift*”, “*670 NEAR:5 solution*”
- To support wildcard queries?
  - “*tayl**”, “**ift*”

Phrase Queries

- *"taylor swift", "670 homework solution"*
- Why might we like to support phrase queries?
  - *"taylor made a swift decision to pivot the project after noticing the early results."*
  - *"the professor teaches both 670 and 698, but only the 698 homework solution was shared on the course website."*

Phrase Queries: One Idea

- Bigram index: Index every consecutive pair of terms in the text as a phrase
  - "*taylor made a swift decision* …"

Vocabulary		
taylor	→	…
made	→	…
…		
taylor made	→	…
made a	→	…
a swift	→	…
swift decision	→	…
…		

- What if the query has three words?
  - Trigram index: Index every consecutive span of three terms in the text as a phrase.
- What if the query has four words?
  - …
- Problems with this strategy?

Instead: Positional Index

- Store the position in the index!
- Document 1 (d1):"*any choose love*"
- Document 2 (d2):"*zebra any love any zebra*"

Vocabulary
any
believe
choose
love
midnight
starring
zebra

Instead: Positional Index

- Store the position in the index!
- Document 1 (d1): "*any choose love*"
- Document 2 (d2): "*zebra any love any zebra*"

Vocabulary		
any	→	d1 (0)
believe		
choose	→	d1 (1)
love	→	d1 (2)
midnight		
starring		
zebra		

Instead: Positional Index

- Store the position in the index!
- Document 1 (d1): "*any choose love*"
- Document 2 (d2): "*zebra any love any zebra*"

Vocabulary				
any	→	d1 (0)	→	d2 (1, 3)
believe				
choose	→	d1 (1)		
love	→	d1 (2)	→	d2 (2)
midnight				
starring				
zebra	→	d2 (0, 4)		

Positional Index: Querying

- Query 1: "*any love*" (phrase)

Vocabulary				
any	→	d1 (0)	→	d2 (1, 3)
love	→	d1 (2)	→	d2 (2)

- Constraint 1: "*any*" and "*love*" should appear in the same document.
- Constraint 2: In this document, position("*love*") = position("*any*") + 1

- Query 2: "*love any zebra*" (phrase)

Vocabulary				
any	→	d1 (0)	→	d2 (1, 3)
love	→	d1 (2)	→	d2 (2)
zebra	→	d2 (0, 4)		

Proximity Queries

- "*taylor NEAR:2 swift*", "*670 NEAR:5 solution*"
- "*NEAR:k*" (or "*/k*"): within *k* words of (on either side)
- Why might we like to support proximity queries?
  - "*Taylor Alison Swift (born December 13, 1989) is an American singer-songwriter. Known for her autobiographical songwriting, artistic versatility, and cultural impact …*"

- Positional index still works!
- Query: "*zebra NEAR:2 love*"

Vocabulary				
love	→	d1 (2)	→	d2 (2)
zebra	→	d2 (0, 4)		

- Constraint 1: "*zebra*" and "*love*" should appear in the same document.
- Constraint 2: In this document, |position("*zebra*") - position("*love*")| ≤ 2.

Wildcard Queries

- "*mid**"
- Find all documents containing any word that begins with "*mid*"
  - "*midnight*", "*midnights*", "*midnoon*", "*midas*", …

Wildcard Queries: One Idea

- “*mid**”
- Suppose we have a binary search tree over our dictionary
  - Find all words in range: “*mid*” <= words < “*mie*”

	Vocabulary								
	any	→	d1	→	d2	→	d5		
	believe	→	d4						
	…								
mid -->	midas	→	d1	→	d5	→	d6	→	d10
	midnight	→	d1	→	d2	→	d8		
	midnoon	→	d7	→	d9				
mie -->	mine	→	d7	→	d9				
	…								
	zebra	→	d2	→	d8				

But we have harder cases

- What about wildcards at the beginning of a word?
  - "**ift*"
  - Find all documents containing any word that ends with "*ift*"
    - "*swift*", "*lift*", "*rift*", …
- What about wildcards at any point of a word?
  - "*ta*r*": "*taylor*", "*tater*", "*tailor*", …
  - "*m*ight*": "*midnight*", "*moonlight*", …
- Ideas?

Permuterm Index

- Use a special end-of-word token, e.g., “$”
  - “*taylor$*”
- Rotate every term:
  - “*taylor$*”, “*aylor$t*”, “*ylor$ta*”, “*lor$tay*”, “*or$tayl*”, “*r$taylo*”, “*$taylor*”

- If we have the query
  - “*ta*r*”
- Rotate the query, so that the “*” is at the end!
  - “*r$ta**”

- Look up in the rotated dictionary. “*taylor*”, “*tater*”, “*tailor*” should all be near each other:
  - “*r$taylor*”, “*r$tater*”, “*r$tailor*”

Summary

- To support phrase queries?
  - *“taylor swift”,“670 homework solution”*
  - Positional index
- To support proximity queries?
  - *“taylor NEAR:2 swift”,“670 NEAR:5 solution”*
  - Positional index
- To support wildcard queries?
  - *“tayl*”,“*ift”*
  - Permuterm Index

Extended Content
(will not appear in quizzes or the exam)

What should be in the index?

- What are the valid tokens to go in our dictionary?

- Input: a bunch of text
  - E.g., "*Welcome to class 670ers!*"
- Output: valid tokens
  - Approach 1: "*Welcome*", "*to*", "*class*", "*670ers!*"
  - Approach 2: "*welcome*", "*to*", "*class*", "*670ers*", "*!*"
  - Approach 3: "*wel*", "*elc*", "*lco*", "*com*", "*ome*", "*to*", "*cla*", …

- Critical step in determining what our users can search for.
  - If a token is not in our index, then our user cannot search for it!

Typical Issues in Tokenization

- Punctuation
  - "*pre-trained*" → "*pre-trained*" or even "*pretrained*" (better than "*pre*", "*trained*")
  - "*U.S.A.*" → "*U.S.A.*" or even "*USA*" (better than "*U*", "*S*", "*A*")
  - "*C.A.T.*" → "*C.A.T.*" (better than "*cat*")
  - "*A&M*" → "*A&M*" (better than "*AM*" or "*A*", "*M*")
- Case
  - "*PageRank*", "*Pagerank*", "*PAGERANK*" → "*pagerank*"
  - "*Apple*", "*Windows*" → ? (depending on the context)
- Domain/Task
  - "*F = ma*" → "*F = ma*" (better than "*F*", "=", "*ma*")
  - "*2%-4%*" → "*2%*", "-", "*4%*"
  - "*CaO + CO2 = CaCO3*" → ? (depending on your task: retrieving the reactants vs. retrieving the equation)

Stemming

- The same word can be used in different forms.
  - “*organize*”, “*organized*”, “*organizes*”, “*organizing*”
- There are families of derivationally related words with similar meanings.
  - “*democracy*”, “*democratic*”, “*democratization*”
- When you search one of these words, you may also want documents containing other words in the set.

- Stemming: Reducing inflectional forms and sometimes derivationally related forms of a word to a common base form

Porter's Algorithm

- Stemming: Reducing inflectional forms and sometimes derivationally related forms of a word to a common base form

- Porter's Algorithm: 5 phases of word reduction applied sequentially
  - Phase 1

Rule			**Example**		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

  - For more information: http://www.tartarus.org/martin/PorterStemmer/

Examples of Stemming

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

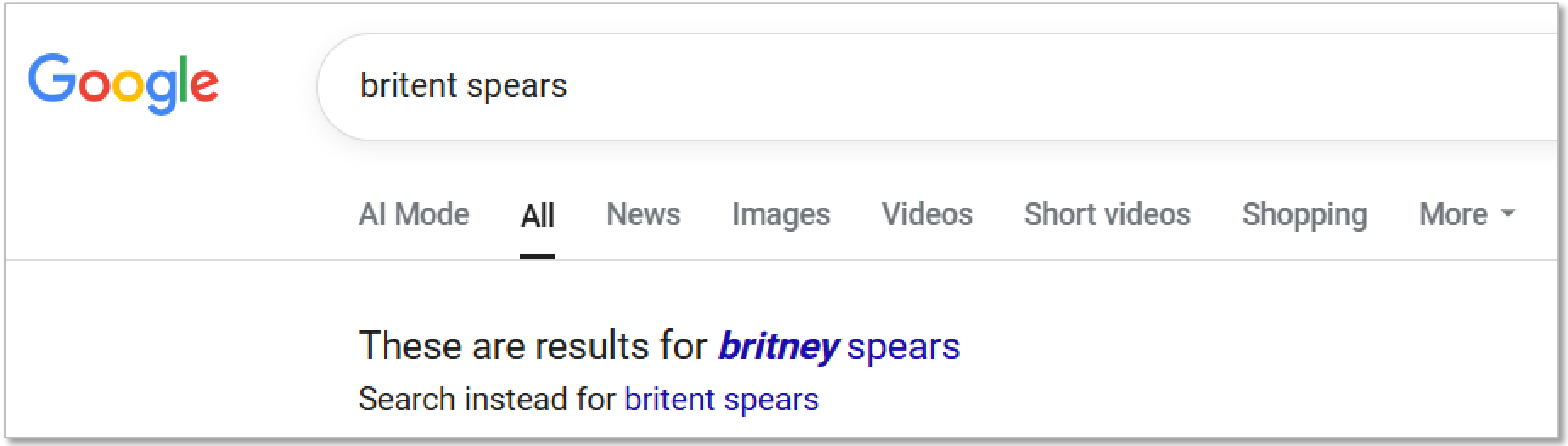
Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Spelling Errors

- Users make spelling mistakes all the time.
- How do we know that?
  - *“britent spears”* → *“britney spears”*



Spelling Correction

- Based on edit distance
- Based on everyone's search logs

https://archive.google/jobs/britney.html

488941 britney spears	29 britent spears	9 brinttany spears	5 brney spears	3 britiy spears	2 brirreny spears
40134 brittany spears	29 brittnany spears	9 britanay spears	5 broitney spears	3 britmeny spears	2 brirtany spears
36315 brittney spears	29 britttany spears	9 britinany spears	5 brotny spears	3 britneeey spears	2 brirttany spears
24342 britany spears	29 btiney spears	9 britn spears	5 bruteny spears	3 britnehy spears	2 brirttney spears
7331 britny spears	26 birttney spears	9 britnew spears	5 btiyney spears	3 britnely spears	2 britain spears
6633 briteny spears	26 breitney spears	9 britneyn spears	5 btrittney spears	3 britnesy spears	2 britane spears
2696 britteny spears	26 brinity spears	9 britrney spears	5 gritney spears	3 britnetty spears	2 britaneny spears
1807 briney spears	26 britenay spears	9 brtiny spears	5 spritney spears	3 britnex spears	2 britania spears
1635 brittny spears	26 britneyt spears	9 brtittney spears	4 bittny spears	3 britneyxxx spears	2 britann spears
1479 brintey spears	26 brittan spears	9 brtny spears	4 bnritney spears	3 britnity spears	2 britanna spears
1479 britanny spears	26 brittne spears	9 brytny spears	4 brandy spears	3 britntey spears	2 britannie spears
1338 britiny spears	26 btittany spears	9 rbitney spears	4 brbritney spears	3 britnyey spears	2 britannt spears
1211 britnet spears	24 beitney spears	8 birtiny spears	4 breatiny spears	3 britterny spears	2 britannu spears
1096 britiney spears	24 birteny spears	8 bithney spears	4 breetney spears	3 brittneey spears	2 britanyl spears
991 britaney spears	24 brightney spears	8 brattany spears	4 bretiney spears	3 brittnney spears	2 britanyt spears
991 britnay spears	24 brintiny spears	8 breitny spears	4 brfitney spears	3 brittnyey spears	2 briteeny spears
811 brithney spears	24 britanty spears	8 breteny spears	4 briattany spears	3 brityen spears	2 britenany spears
811 brtiney spears	24 britenny spears	8 brightny spears	4 brieteny spears	3 briytney spears	2 britenet spears
664 birtney spears	24 britini spears	8 brintay spears	4 briety spears	3 brltney spears	2 briteniy spears
664 brintney spears	24 britnwy spears	8 brinttey spears	4 briitny spears	3 broteny spears	2 britenys spears
664 briteney spears	24 brittni spears	8 briotney spears	4 briittany spears	3 brtaney spears	2 britianey spears

Thank You!

Course Website: https://yuzhang-teaching.github.io/CSCE670-F25.html