Overview of topics in next 2 weeks (last few lectures):

- Vector DBs → Project 4 (shorter)
- Retrieval
- RAG (using LLMs for Knowledge-Intensive Tasks)
- * Compound AI systems
- * If time: Video systems + model debugging

Today: Basics of vector DBs, intro to retrieval

* First off: what is retrieval? how does vector similarity / vector DB fit in?

→ "Information Retrieval" = SEARCH

↳ "finding material that fulfills an information need from within a large collection of unstructured documents"

large scale search !!!

↳ text, media, etc.; not necessarily structured info like in a database

- There are different types of IR Tasks:

- in most types, the user will

specify a "query" - but this could be ambiguous - so an IR system may actually augment the query provided w/ knowledge of TASK and USER (consider google search):

Expression of Information Need	Potential Query	Potential Collection
Find related literature	The full text of the BERT paper	ACL anthology; arXiv CL
Recommend me a TV show to watch	[no explicit query!]	Netflix shows
Find every relevant patent	Boolean query with technical terms	U.S. Patents
Buy a new laptop	Short conversation: system asks questions to ascertain your criteria	E-commerce platforms

Omar Khattab, Stanford CS224U Lectures On Information Retrieval

*all IR tasks have unique challenges:

- consider large-scale web search vs. search within slack:

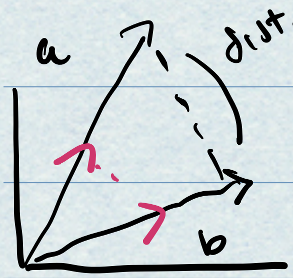
LARGE-SCALE web search	SLACK:
- popular "head queries"	✗
- redundant documents on common topics	✗
- explicit hyperlinks between documents	✗
- LARGE-scale	SMALL-scale

- VECTOR Databases:
  - store "representation" of data that can later be useful for comparing dif. pieces of data
  - can think of vector similarity as a TECHNIQUE used in retrieval systems - but vector DBs themselves are used in arange of apps past IR-related tasks
    - ↳ e.g., find similar images
  - vector DBs also support CRUD: will change over time

    create read update delete
    - ↳ IR indexes typically don't change after they are built

- VECTOR SIMILARITY METRICS:
  1) Euclidean Distance

  $$d(a,b) = \sqrt{(a_1-b_1)^2+(a_2-b_2)^2 \dots (a_n-b_n)^2}$$



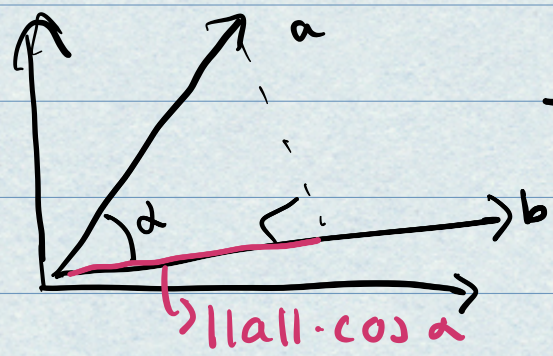
  - considers magnitude <u>And</u> direction
  - sensitive to magnitude — larger vectors that are close together have larger distance than same vectors scaled down

2) Dot Product similarity:

$$a \cdot b = \sum_{i=1}^{n} a_i \cdot b_i$$

↳ can also be expressed as:

$$a \cdot b = \|a\| \, \|b\| \cos \alpha$$



→ Dot product PROJECTS one vector over the other

- affected by BOTH length and direction

→ DOTPROD > 0 if $\alpha < 90°$

< 0 if $\alpha > 90°$

$= 0$ if vectors are perpendicular

3) cosine similarity → Just angle

$$sim(a,b) = \frac{a \cdot b}{\|a\| \cdot \|b\|}$$ → IGNORE DISTANCE

* in general, its a good idea to use same similarity metric (for inference) used to train embedding model

- TRADEOFFS in VECTOR SIMILARITY SEARCH:

* formal problem: given a set

of base vectors and a query vector, which of the base vectors is most similar to the query vector?

↳ assume some similarity metric

↳ could also consider top-k, but don't worry about that for now

- TRADEOFFS:

1. ACCURACY - don't necessarily need to return *exact* most simil

2. SPEED - what is inference cost?

3. MEMORY - past memory needed to store actual base vectors, how much memory are we storing?

*behind every vector DB is an INDEX - index choice trades off on

(Brief) overview of a few different index options:

1. Flat

- store vectors as list
- given a query, iterate through all base vectors and find most similar

* Discuss: what is speed, memory, accuracy tradeoff?

- How can we make search faster?
  - A: reduce size of vectors so any one similarity search takes less time
  - B: Store extra metadata ahead of time, so we don't have to search EVERY vector

2. LSH (Locality-sensitive Hashing)

  1. hash the sample (multiple times)
     - if a base vector has been hashed to the same value Atleast once - consider it a "candidate"
  2. once we have candidates, do normal similarity search

* try to use hash function that

MAXIMIZES collisions (ideally for similar inputs)

- we'll go over 1 particular way to do this, but there are many dif. types of hash-based indices!
  - 3 actual steps:
    1. convert vectors to a sparse representation
    2. use minhashing to create signatures for each vectors
    3. signatures passed to LSH to find candidate pairs

→ step 1: create sparse representation

- consider all examples (e.g. text)
- create "vocab" by sliding over all text at some window

"flying fish flew by the space station":

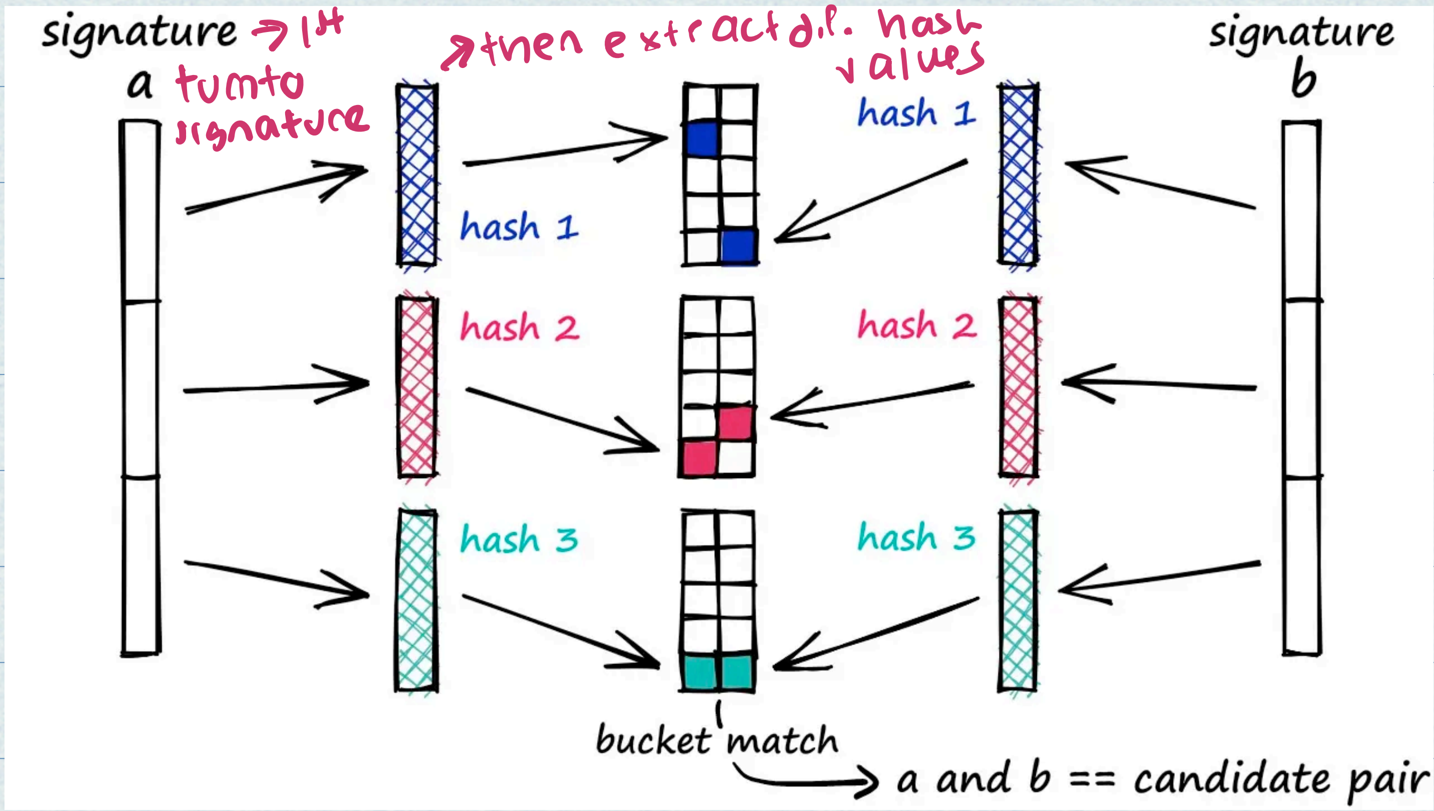
		shuffle	zero-vector	one-hot
fl		ly	0	1
ly		il	0	0
yi	→	ti	0	1
...		...	⋮	...
ti		or	0	0
io		dy	0	0

on sp 0 1

(+ more for other examples)

→step 2: turn sparse representation into MULTIPLE hash values

↳we will ignore details



https://www.pinecone.io/learn/series/faiss/locality-sensitive-hashing/

→step 3:

- a base vector is a candidate if it has ANY common hashes

*can do work of building signatures /hashes of base vectors offline

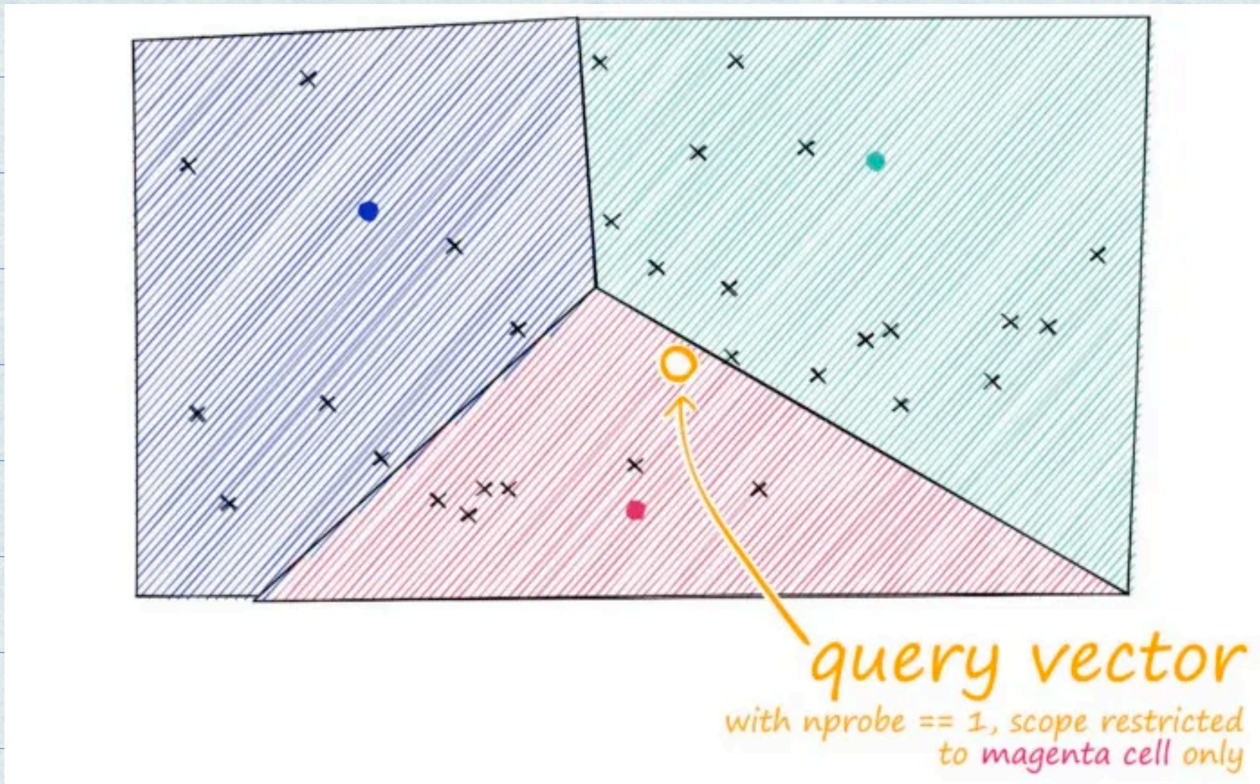
↳add memory overhead to store

tms

* at runtime, run signature creation/ hashing on query vector, and compare to all candidates

3. Inverted File Indices (Proj. 4)

→ cluster data into "cells"

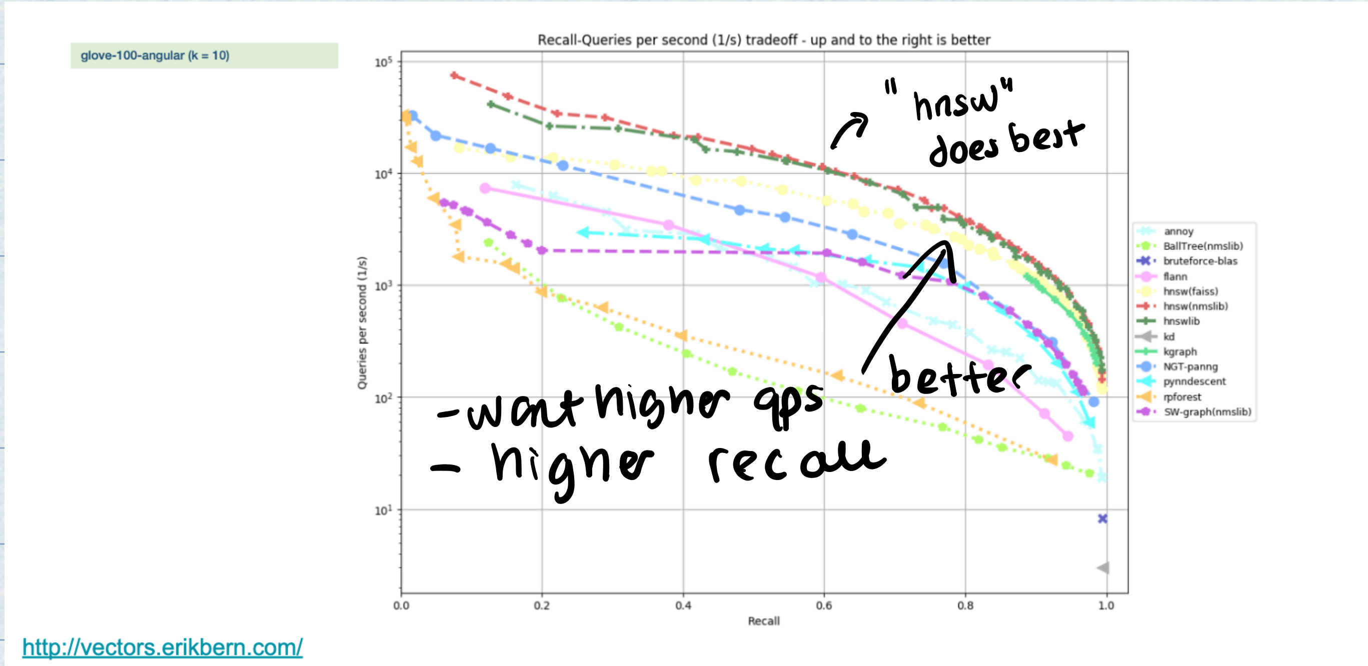


two params:

- nprobe - how many cells do I search
- nlist - how many cells do I create?

* Discussion: how might varying nprobe and nlist change accuracy vs. speed?

4. Graph-Based Index / HNSW



cs229s 24 slides,
Liana Patel

- general paradigm of HNSW : "hierarchical navigable small worlds"
  1) construct a "proximity graph"
     - vectors = nodes
     - edges connect nearby vectors
  2) search /Traverse index
     - use greedy search from pre-defined entry point

- how do we build graph?
  - for fast search - want any two nodes to be connected by short path

*stay tuned for more on thursday!