[Day 2] Unit 2 - "Embeddings & Vector Stores / Databases"

Embeddings

- "cheat sheet" for machines to understand meaning of text, images & other data.
  - help make sense of massive data.
- low dimensional numerical representations of "anything"
  - designed to capture meaning & relationships between different pieces of data.
    - ↳ compact vectors preserve semantic information
    - → map complex data into a simpler space
- ★ Incredibl. efficient for all types of data, save storage space, makes poss. to find patterns/rel.

Semantic Rel.

- embeddings capture how similar/diff two pieces of data are (ex: King ⇄ Queen ⟶ bike)
  - encode meaning, large scale processing

Applications:

- Retrieval & Recommendation: precompute embedding, query is converted to embedding, find nearest embedding
  - retrieves & rec. semantically similar embeddings.
  - recommend: find content similar embedding to what user prev. liked/interact.

Joint-Embedding

- Handle multimodal data (pics, videos, text)
  - *allow for more nuanced understanding

Eval. emb.

- see how well they retrieve relevant data, & filter out irrelevant
- Precision: amount of retrieved items that are relevant.
- Recall: proportion of all relevant items you managed to find.

@ k=5; how many top 5 pred. are correct.

Normalized discount. cumulative gain.

- gives higher scores when relevant items are @ top of results.
  - higher = better

BEIR / MTEB

- standardized collections of data sets & tasks, allow evaluate & compare different emb. models (fair & consistent)

Choosing Emb. Model

- key factors: model size, embedding dimensionality, latency/cost.

Retrieval aug. gen.

- using emb. to make lang. models "smarter", use embedding to find rel. info from know. base, use info to boost prompts given to lang model
  - ↑ accurate & ↑ relevant responses

2-Stage process

1st
- Creat index, generate emb. for chunks using a doc encoder, storing emb. in vector DB.

2nd
- Query processing stage: take user q, turn into emb., perform similarity search in vector DB to find chunks closest to q

Challenges

- Need for labeled data. for training ($ & time cons.)
  → ex sol: use LLM to generate synthetic question & doc pairs

Types of Embeddings

- Text. allow to rep. text as dense numerical vectors
  *powerful for NLP tasks
- Token/Word: Breakdown text string into smaller meaningful units (each unique gets unique # ID)
- One-Hot encoding: rep token IDs as Binary vectors
  * dont capture semantics
- word emb: dense, fixed length vectors aim to capture meaning of indiv. words
- Word2vec: meaning of word is determined by words around it
  -architech: CBOW & skip gram → better for infrequent words & smaller DS
  trains faster (common words)
  - limits: small window of context
- Co-occurrence matrix: cont. how often each word appears in context of every other word in DS
- doc emb: emb. large pieces of text
  - bag of words models (LSA: look for underlying themes, LDA: probab. approach) (TF IDF: Stat. method that weigh words on how often they appear)
- Visual emb: training convolutional, neural networks or vision trans.
- Multi-modal emb: combines text & image emb.
- Graph emb: rep. objects and relationships within a network.
  - algos: deepwalk, node to vec line.

Training emb

- Use dual encoder architech, trained w/ (constrative loss)
  → pulls emb. of similar data closer together & repelling diff.
  → 2 stages: pretraining & fine tuning

Vector search

- Finding items based on their meaning not just matching kw
  - Use algs like approx. nearest neighbor.

ANN techs
- locality sensitive hashing, tree based methods
  - HSNW go-to algo. handles new insertion of vectors efficient.
  - SCANN: partitioning

Vector DB/stores
- doc emb. → index vector → embed. query → search for similar

Applications
- info retrieval, recomend., semantic text similarity, classification, clustring

[Live Stream]
- Emb. advancement: using LLMs to speed up emb. training
  - LLMs: train training data sets. & data curation.
  - matrioshka emb:
- Alloy DB
- Efficient techs for vector search: benchmarks for use can
  → RAG search: combing key word & embeddings (hybrid search)
  → filtering and re-ranking.
- Upgrading all embeding models when using new ones.
  → upgrad embeddings as well.
  * align search: efficient way to match prev. emb to new.
    ↳ biggest pain to upgrade DB.
- RAG & measuring drift, focus more on recall > retrieval
  *bridge gap between index and new context (live data)
- Freshness of data as a new eval. metric.

Pop quiz:
1) Application emb are NOT best suited for
   → simple rule based system
2) Adv. of scan > ANN search algs?
   → designed for high-dim. data, & excellent acc/latency tradeoffs
3) weakness of BOW for gen. embeddings?
   → ignore word ordering & sem. meaning
4) Challenge when using emb for search? address?
   → emb might not capture literal meaning
5) Adv. of LSH for vector search
   → reduces search space by reducing dims to buckets

[codelabs]

Doc Q+A w/ RAG using chroma

- limits of LLMs: only "know" info. trained on, limited input context windows → address by: RAG
- RAG (Retrieval augmented Generation) stages:
  1) Indexing 2) Retrieval 3) Generation
  *allows for providing info that model hasn't seen b4
- Data → chnk emb. DB w/ ChromaDB → retrieval (find relevant docs) → Augmented gen (answer Q's)

Embeddings and Similarity Scores

- Calc. similarity scores of 2 emb. vectors: $u^T v$
  → Range: 0 ⇒ dissimilar, Range=1 = comp. similar
  → calc by matrix self-mult: sim = df @ df.T

Classifying emb. w/ Keras + the Gemini API

- Dataset + Embeddings (comp. on @ a time) → Build classification model. → train model → evaluate model performance,