RavenDB

Vector Search

Evolution of Searching

RavenDB

- Exact matching
  “Car” == “Car”

- Partial/Wildcard matching
  “Car*” == “Carpet”

- Full-text matching
  “My car is red” => [“my”, “car”, “red”] == “car”

- Semantic matching
  “Signature red car” => “Ferrari”

Semantic similarity- how?

- Take Phrase1
- Convert it to NumericalRepresentation1
- Take Phrase2
- Convert it to NumericalRepresentation2

- Compute distance between NR1 and NR2
- Distance == Similarity

How to represent words numerically?

- “Beach Towel”

beach	0.91
sun	0.87
water	0.78
sand	0.84
swimwear	0.72
vacation	0.76
sunscreen	0.70
summer	0.86
waves	0.75
sunbathing	0.82
surf	0.68
pool	0.73
heat	0.65
resort	0.77
winter	0.24
snow	0.19
wind	0.30
skiing	0.16
jacket	0.22

How to represent words numerically?

“Beach Towel”	
beach	0.91
sun	0.87
water	0.78
sand	0.84
swimwear	0.72
vacation	0.76
sunscreen	0.70
summer	0.86
waves	0.75
sunbathing	0.82
surf	0.68
pool	0.73
heat	0.65
resort	0.77
winter	0.24
snow	0.19
wind	0.30
skiing	0.16
jacket	0.22

“Sunglasses”	
beach	0.84
sun	0.93
water	0.72
sand	0.75
swimwear	0.70
vacation	0.80
sunscreen	0.78
summer	0.89
waves	0.70
sunbathing	0.82
surf	0.66
pool	0.73
heat	0.76
resort	0.78
winter	0.28
snow	0.34
wind	0.42
skiing	0.38
jacket	0.25

How to compute similarity?

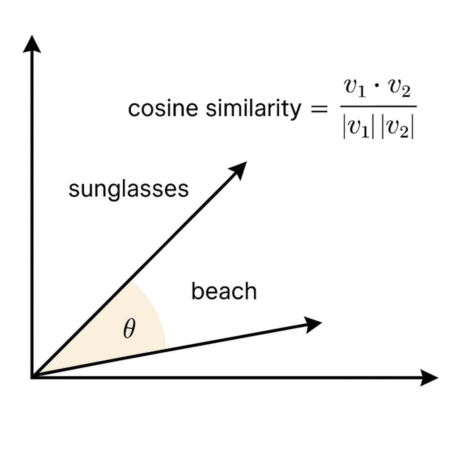
1. Represent numbers as vectors - “vector embeddings”

   sunglasses → $\mathbf{v}_1$ = `[0.84, 0.93, ..., 0.25]`
   beach → $\mathbf{v}_2$ = `[0.91, 0.87, ..., 0.22]`

2. Compute “Cosine Similarity”

   1.0 → perfectly similar (same direction)
   0.0 → unrelated (orthogonal)

   0.81 = 81%



How to automate this?

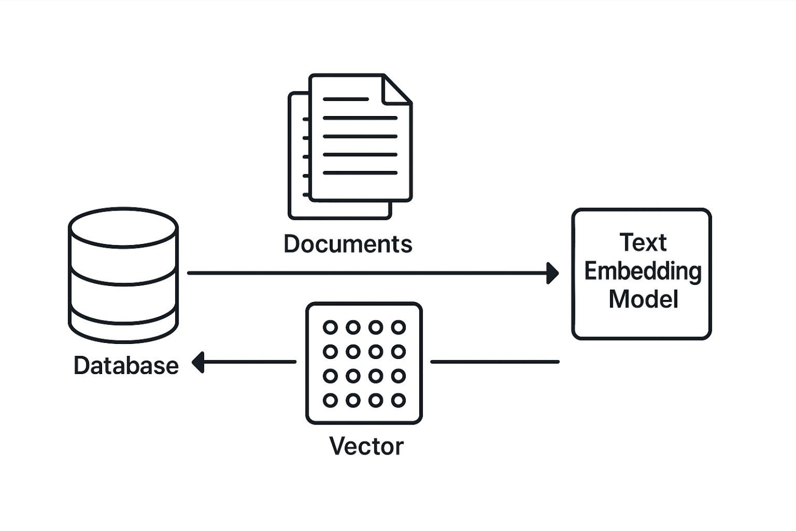
- AI, of course!
- “Text Embedding Models” with various “vector dimensionality”

Word2Vec (Google News)	- 300	- one of earliest, not context aware
FastText (Facebook)	- 300	- good for misspelled words
Universal Sentence Encoder (Google)	- 512	- semantic tasks and question answering
BERT	- 768	
OpenAI Ada v2	- 1536	

- We have no idea what attributes are :)
- NOT LLMs!

Vector Search in RavenDB

- Available in v7
- Embedded bge-micro-v2 (384) for fast prototyping
- AI Tasks - Embeddings Generation - support for external models



Secondary aspects

- Caching
    - Reduce latency
    - Reduce cost

      OpenAI text-embedding-3-large : $0.00013 / 1k tokens

- Compression

      OpenAI text-embedding-3-small - 1536
          100M embeddings - 572 GB
          250M embeddings - 1.430 GB

      OpenAI text-embedding-3-large - 3072
          100M embeddings - 1.144 GB
          250M embeddings - 2.861 GB

    - Quantization
        - Single (no quantization)
        - Int8 (float32 -> int8, scaling)
            - fast, moderate accuracy loss, 4x smaller
        - Binary (float32 -> 1 binary bit)   [0.2, -0.5, 1.3] -> [1, 0, 1]   (threshold is 0)
            - exploits overparameterization, use on larger models, e.g. over 1024
            - extremely fast, high accuracy loss, 32x smaller

Not just search!

RavenDB

- Clustering
  - Group similar items together
  - Detect emerging trends
  - Predict missing relationships
- Recommendation systems
  - Find similar items
- Anomaly detection
  - Vectors far from normal distribution
- Content deduplication
  - Detect nearly-identical items
- User Profiling / Behavior Modelling