AI mit Oracle

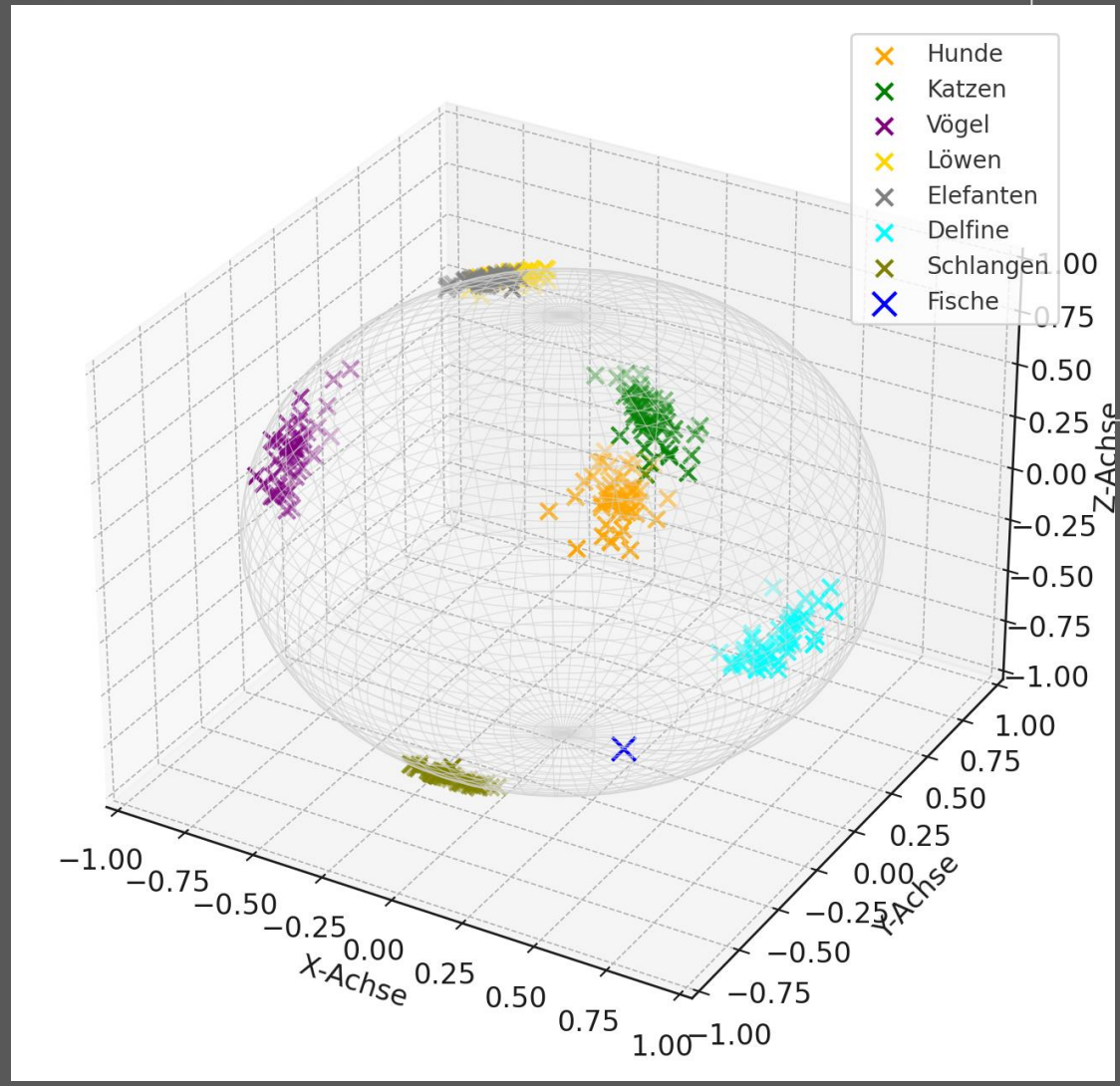
Wir bauen einen RAG-basierten Chat

Teil 3.2 – RAG & Chat mit KI: Vektorsuche

graef.

Vector-Distance als Ähnlichkeitsmerkmal

Grundlage: Ähnliche Texte erzeugen ähnliche Vektoren
➔ ähnliche Texte haben einen kurzen Vektorabstand



RAG – Kontextunterstützung des Chats: Vektorsuche

- Grundsätzliche Vorüberlegungen
  - Generell: Vergleich von Vektoren geht nur mit gleichartigen Vektoren.
    Beide Vektoren müssen mit dem identischen Verfahren erstellt worden sein!
  - Vektorisierung an zwei Stellen nötig:
    1. Dokumentenbasis
    2. Anfrage bzw. Prompt

  → Vektorisierung muss aus PL/SQL heraus durchführbar sein!

- Lösungsansätze:
  - Externer Service (z.B. ChatGPT)
  - Eigener REST-Service mit Python & Docker
  - ONNX-Integration in Oracle

- Abstandsbestimmung zur Suche nach Best-Matches

graef.

Vektorisierung mit ChatGPT

Implementation:

- Oracle-Credential für OpenAI (schon in Teil 1 eingerichtet)
- Befehl: *dbms_vector.utl_to_embedding*
- JSON-Objekt für Aufrufparameter:

```
{
  "provider": "openai",
  "credential_name": „<Name des Credentials>",
  "url": "https://api.openai.com/v1/embeddings",
  "model": „<Modellname>"
}
```

- Optional: Vector-Caching zur Kostenreduzierung

graef.

Vektorisierung als REST Service

Implementation in Zwei Teilen:

1. Python REST Service auf Docker im Firmennetz
   - Bibliotheken:
     - flask (Webservice)
     - sentence_transformers (Vector-Modelle)
     - numpy
     - json
     - urllib
   - Vector-Modelle:
     - all-MiniLM-L6-v2 (Dimension=384)
     - paraphrase-multilingual-mpnet-base-v2 (Dimension=768)
     - als Endpoints des REST-Services
2. PL/SQL Datenbankseitig:
   - JSON Objekt zur Kommunikation mit REST-Service
   - Verwendung von UTL_HTTP (analog der Kommunikation mit der Open-AI-API aus Teil 1)

graef.

Vektorisierung mit ONNX Modellen

Implementation:

- Installation nach https://blogs.oracle.com/machinelearning/post/use-our-prebuilt-onnx-model-now-available-for-embedding-generation-in-oracle-database-23ai

- Verkapselung des Aufrufs von VECTOR_EMBEDDING(<Modellname> USING ‘<Text>‘ AS data), um flexibel verschiedene Modelle nutzen zu können, ohne neu kompilieren zu müssen.

graef.

Vergleich der Verfahren

VM_PK	VM_STRNAME	VM_STRCODE	TYP	Dauer (Sek.)	DIM	Kommentar
1	OpenAI: text-embedding-3-small	text-embedding-3-small	OpenAI	1,134	1536	
2	OpenAI: text-embedding-3-large	text-embedding-3-large	OpenAI	1,164	3072	from_vector(...) not working
3	OpenAI: text-embedding-ada-002	text-embedding-ada-002	OpenAI	0,761	1536	
5	paraphrase-multilingual-mpnet-base-v2	paraphrase-multilingual-mpnet-base-v2	REST	0,139	768	
6	all-MiniLM-L6-v2	all-MiniLM-L6-v2	REST	0,07	384	
7	all-MiniLM-L6-v2.onnx	common.MINILM_L6_V2	ONNX	21,215	384	Instabil auf 23AI free / Freeze
8	all_MINILM_L12_V2.onnx	common.MINILM_L12_V2	ONNX	22,771	384	Instabil auf 23AI free / Freeze

Ergebnis:

- **Open AI ist hochdimensional, kann ggf. Vorteile bei längeren Texten bei der Treffergenauigkeit bieten, kostet aber Geld.**
- **ONNX: Einfach zu implementieren, benötigt keine extra Infrastruktur, ist aber langsam.**
  Es wäre spannend zu sehen, wie sich das Timing verändert, wen die Limitationen der Free-Version nicht mehr greifen, insbesondere die CPU-Einschränkungen entfallen.
- **Eigener REST-Server: sehr schnell, insbesondere mit ONNX vergleichbar, da identische Verfahren.**
  Benötigt aber extra Server mit Docker. (Ggf. mit CUDA noch schneller.)

graef

Fazit

- Verwendbarkeit
  - \+ Einzeldaten aus Dokumenten extrahieren
  - – Suchen, die ALLE Stellen erfordern / Zusammenfassungen
    → Verschiedene Modi implementieren?
- Problematik
  - Haluzinationen bei wenig passenden Daten
  - kein Filter, nur Priorisierung der Treffer

→ Beschränkung auf feste Anzahl an Treffern kann ggf. nicht optimal sein

Graef Computer GmbH · Fallgatter 5 · 44369 Dortmund
Telefon: +49 231 222 429 - 99 · info@graef.com