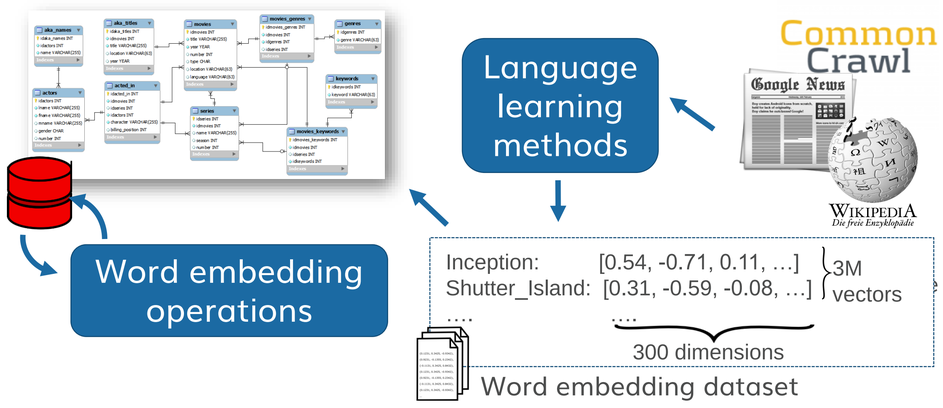
Explore FREDDY: Fast Word Embeddings in Database Systems

Dresden Database Systems Group

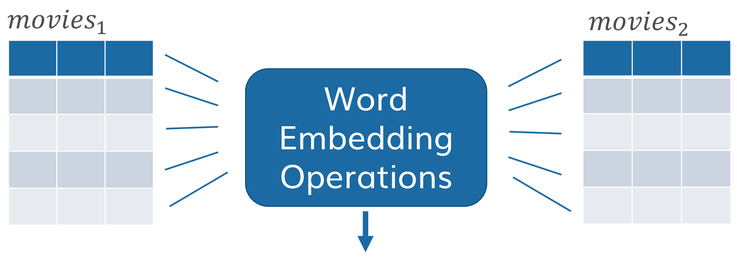
Word Embeddings in Database Systems



Language learning methods (**word2vec**, **fastText)** extract semantic word relations → **Word Embeddings**

Importing word embeddings in a relational database system
→ Enables inductive reasoning on text values

SELECT m.title, t.word, t.squaredistance
FROM movies **AS** m, most_similar(m.title, (SELECT title FROM movies)) **AS** t



Results:
Inception | Shutter Island
…

Similarity of tokens word vectors corresponds to:
- High cosine similarity
- Low Euclidian distance

Challenges

- Integrate operations in SQL
- Sufficient performance to execute multiple operations for one query during runtime → **approximated nearest neighbor search**
- Accomplish different demands on precision and execution time

Word Embedding Operations

SELECT keyword
FROM keywords
ORDER BY cosine_similarity('comedy', keyword)
→ comedy, sitcom, dramedy, comic, satire, …

Cosine_similarity(varchar t1, varchar t2):
Calculating the cosine similarity of two token

SELECT analogy ('Godfather', 'Francis_Ford_Coppola', m.title)
FROM movies **AS** m
Inception → Christopher Nolan

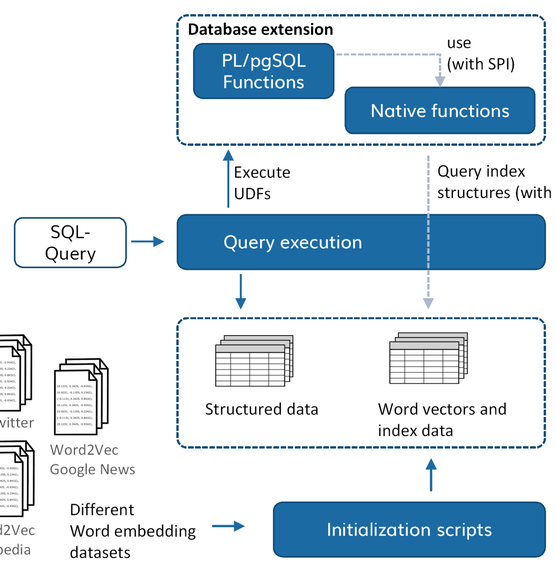
analogy(varchar t1, varchar t2, varchar t3): answer analogy queries

SELECT m.title, t.term, t.score
FROM movies AS kNN(m.title, 3) **AS** t
ORDER BY m.title **ASC**, t.score **DESC**
→ Godfather | {Scarface, Goodfellas, Untouchables}

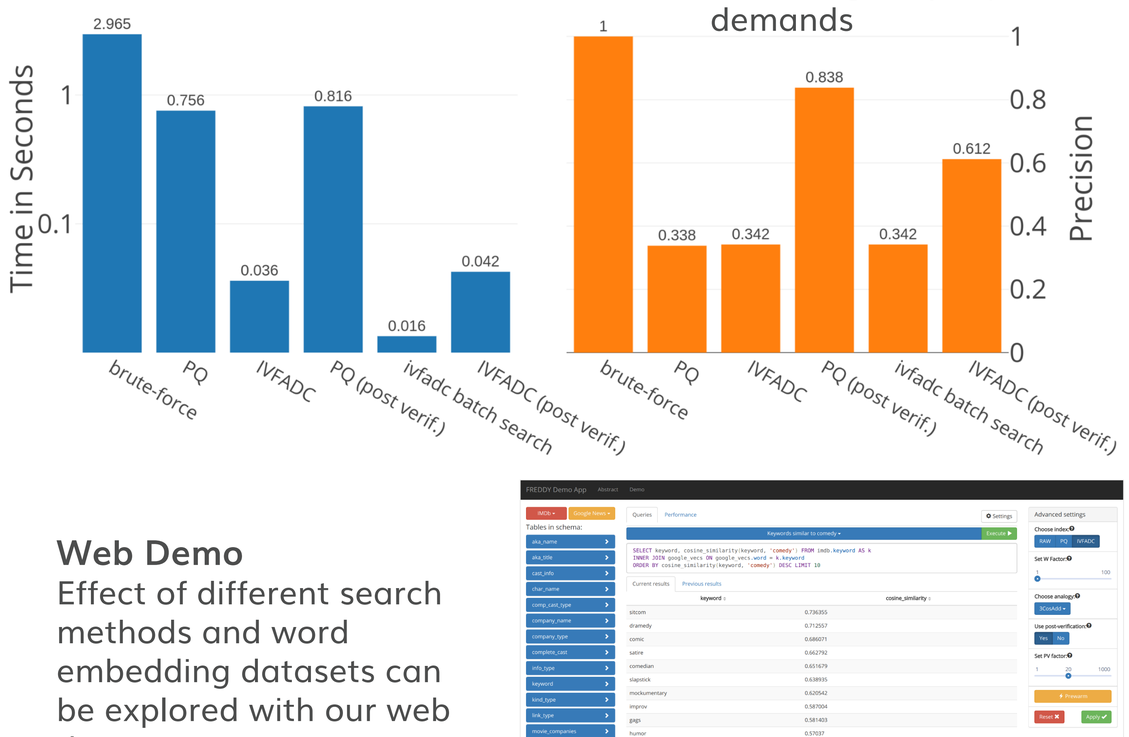
kNN(varchar t, int k): search for *k* most similar tokens in a word embedding dataset

System Architecture

- Extended Postgres Database System: **FREDDY**
- Extension with novel Word-Embedding Operations (**UDFs**)
- Index structures of word embeddings as database relations
- Different search methods for different operations (non-exhaustive, exhaustive and exact search) based on **product quantization**



Search Methods



Post verification and **batch-wise** execution according to query demands

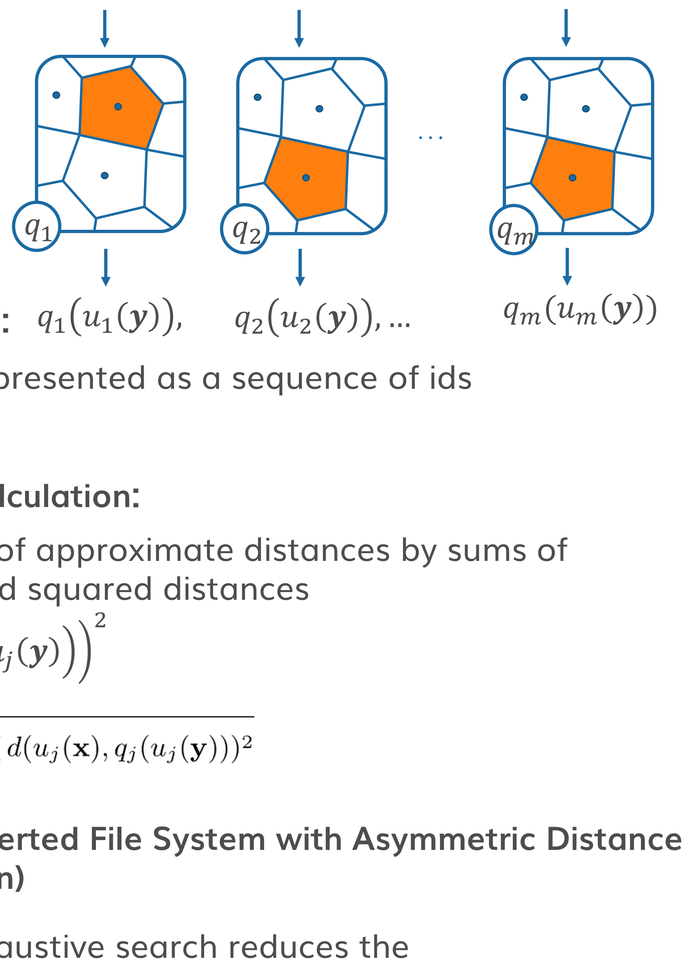
Web Demo
Effect of different search methods and word embedding datasets can be explored with our web demo

Product Quantization for Fast Similarity Search

Idea: Reduce the computation time of the Euclidean square distance through an approximation by a sum of precomputed distances

Quantizer functions q assign subvectors $u_i(\boldsymbol{y})$ to centroid $\{\boldsymbol{c}_1, \dots, \boldsymbol{c}_k\}$ $\quad q : \mathbb{R}^d \rightarrow \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_k\}$

$$\boldsymbol{y} = \underbrace{\{y_1, \dots, y_d,}_{u_1(\boldsymbol{y})} \underbrace{y_{d+1}, \dots, y_{2d}}_{u_2(\boldsymbol{y})} \dots \underbrace{y_{(n-d)+1}, \dots y_n\}}_{u_m(\boldsymbol{y})}$$



Product Quantization: $q_1(u_1(\boldsymbol{y})), \quad q_2(u_2(\boldsymbol{y})), \dots \quad q_m(u_m(\boldsymbol{y}))$

→ Can be represented as a sequence of ids $\{1, \dots k\}$

Distance Calculation:

Calculation of approximate distances by sums of precomputed squared distances

$$\mathrm{d}\left(u_j(\boldsymbol{x}), q_j\left(u_j(\boldsymbol{y})\right)\right)^2$$

$$\hat{d}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_j d(u_j(\mathbf{x}), q_j(u_j(\mathbf{y})))^2}$$

IVFADC (Inverted File System with Asymmetric Distance Computation)

- Non-exhaustive search reduces the amount of distance computations
- **But:** Not applicable for all operations

TECHNISCHE UNIVERSITÄT DRESDEN

Contact: michael.guenther@tu-dresden.de https://wwwdb.inf.tu-dresden.de/research-projects/freddy/