Indexing for github-semantic-search

Compiled by D.Gueorguiev, from Damien Beneviste’s lecture material, 10/5/2025

Indexing happens in the backend, in module [backend/app/indexing/indexer.py](https://github.com/dimitarpg13/github-semantic-search/blob/main/backend-orig/app/indexing/indexer.py).

We start with some parsed data ( code, markdown ) and we need to derive a vector of it.

The vector representation needs to be captured by some metric e.g. cosine similarity, Euclidean distance, norm, etc.

A diagram of a keyword and a keyword

AI-generated content may be incorrect.

Figure : depicted are the 3 possible ways to index the data

The first way to project the parsed code into a vector representation is via dense vector representation. For obtaining this dense vector representation of the code fragment we are going to use LLM.

Besides using dense representation for vector embeddings we can use sparse representations which are useful in capturing specific tokens. Thus, the sparse representation is used to augment the dense representation of vector embeddings. In the dense representation we get a vector for the whole sentence while in the sparse representation we get vectors capturing specific tokens semantics. BM25 provides us with sparse representation of the text. SPLADE ([see [4]) is using LLM to extract sparse representation which is more flexible in terms of capturing values for the different tokens; it also captures synonyms gracefully.

On the right side of the Figure above we see the organization of the indexed data in the DB. We have metadata – strings we can use to filter further our data.

A diagram of a fashion model

AI-generated content may be incorrect.

Figure : Vector Representation with causal language models

We will use LLM to generate embeddings (dense vector representation) for our data as shown on the Figure above.

A diagram of a graph

AI-generated content may be incorrect.

Figure: Encoder Model for SPLADE

# References

[1] [Foundations of Vector Retrieval, S. Bruch, 2024](https://github.com/dimitarpg13/vector_db_intro/blob/main/articles/Foundations_of_Vector_Retrieval_Bruch_2024.pdf)

[2] [Natural Language Processing for Semantic Search, Pinecone, James Briggs](https://www.pinecone.io/learn/series/nlp/)

[3] [Dense Vectors: Capturing Meaning with Code, Chapter 1 of Pinecone online course, James Briggs](https://www.pinecone.io/learn/series/nlp/dense-vector-embeddings-nlp/)

[4] [SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking, T. Formal et al, 2021](https://github.com/dimitarpg13/vector_db_intro/blob/main/articles/Splade-Sparse_lexical_and_expansion_model_for_first_stage_ranking_Formal_2021.pdf)

[5] [An Approximate Algorithm for Maximum Inner Product Search over Steaming Sparse Vectors, S. Bruch et al, Pinecone, 2023](https://github.com/dimitarpg13/vector_db_intro/blob/main/articles/An_Approximate_Algorithm_for_Maximum_Inner_Product_Search_over_Streaming_Sparse_Vectors_Bruch_2023.pdf)

# Appendix

## Sparse vs Dense Representations of Vector Embeddings

Sparse embeddings are a type of embedding where the majority of values in the vector are zeros. These embeddings are generally high-dimensional, with most dimensions inactive or zero. Sparse embeddings were among the earliest forms of word representation in NLP, seen in techniques like one-hot encoding and bag-of-words models.

Key Characteristics of Sparse Embeddings

-High Dimensionality: involve vectors with thousands and even millions of dimensions.

-Locality: represent each word independently; there is little or no overlap , shared information between embeddings for different words

-Interpretability: each dimension usually represents specific feature e.g. presence or absence of specific word

Techniques involving sparse embeddings

-One-hot encoding: a simple form of sparse embedding where each word in the vocabulary is represented as a vector composed of single-unit in the position of the word with rest filled with zeros

-Bag-of-Words (BoW): each document is represented as a sparse vector, with each dimension corresponding to the count of particular word in a document

-Term Frequency-Inverse Document Frequency (TD-IDF): enhances BoW by assigning weights to words based on their frequency in a document relative to their frequency across all documents. The resulting vectors are still sparse but more informative than simple BoW vectors

Advantages of sparse embeddings

-Interpretability: each dimension corresponds to a specific feature or word

-Exact Representation: