Kaggle | Google

5-Day Gen AI Intensive Course with Google 2025

Whitepaper Companion Podcast (Notes): Embeddings and Vector Stores

Introduction to Embeddings and Vector Stores

- Embeddings are low-dimensional numerical representations of data, designed to capture underlying meaning and relationships[1]. They transform heterogeneous data into a unified vector representation[1].
- They represent data like text, images, and audio as compact vectors, preserving essential semantic information and making it easier to process and compare large amounts of data[1].
- **Analogy**: Embeddings map complex data into a simpler space, similar to how latitude and longitude pinpoint locations on Earth[1].

Importance of Embeddings

- **Efficiency**: Embeddings represent various data types (text, audio, images, video, structured data) efficiently[1].
- **Pattern Recognition**: They facilitate finding patterns and relationships in data that would be difficult to spot otherwise[1].
- **Semantic Relationships**: Embeddings capture how similar or different data pieces are. For example, in an embedding space, "king" is closer to "queen" than to "bicycle"[1].

Applications of Embeddings

- **Retrieval**: Used in search systems like Google. Web pages are pre-computed into embeddings. A search query is converted into an embedding, and the system finds web pages with the nearest neighbor embeddings[1].

- **Recommendation Systems**: Items or content with similar embeddings to a user's past interactions are recommended[1].
- **Joint Embeddings**: Handle multimodal data (text and images) by mapping different data types into a common embedding space[1].

Evaluating Embedding Effectiveness

- **Relevance**: Assessed by how well embeddings retrieve relevant items and filter out irrelevant ones[1].
- **Metrics**:
    - **Precision**: Measures how many of the retrieved items are relevant[1].
    - **Recall**: Measures the proportion of all relevant items that are retrieved[1].
    - **Precision at K & Recall at K**: Focus on the top K results[1].
    - **Normalized Discounted Cumulative Gain (NDCG)**: Gives higher scores when the most relevant items are at the top of the results list[1].
- **Benchmarks**: Standardized collections of datasets and tasks to evaluate and compare different embedding models (e.g., BEIR, MTEB)[1].
- **Libraries**: Established libraries like TEREVAL, TRank, or PyTerrier can be used for evaluation[1].
- **Practical Considerations**: Model size, embedding dimensionality, latency, and cost[1].

Retrieval Augmented Generation (RAG)

- Embeddings enhance language models by finding relevant information from a knowledge base to boost prompts[1].
- **Process**:
    a. **Index Creation**: Documents are broken into chunks, embeddings are generated for each chunk using a document encoder, and embeddings are stored in a vector database[1].

b. **Query Processing**: A user's question is converted into an embedding using a query encoder, and a similarity search is performed in the vector database to find the closest chunks[1].

- **Importance of Speed**: Efficient vector databases are crucial for quick query processing[1].
- **Progress in Embedding Models**: Significant improvements have been made, such as Google's embeddings, where the average BEIR score jumped from 10.6 to 55.7[1].
- **Code Snippet**: The white paper includes a code example using the NF Corpus dataset[1].

Types of Embeddings

- Categorized based on the type of data[1].
  - **Text Embeddings**: Represent words, sentences, paragraphs, or entire documents as numerical vectors[1].
    - **Tokenization**: Breaking down text into smaller units (tokens)[1].
    - **One-Hot Encoding**: Representing token IDs as binary vectors[1].
    - **Word Embeddings**: Capture the meaning of individual words and their semantic relationships[1].
      - **Word2Vec**: Core principle is that a word is known by the company it keeps (context)[1].
        - **CBOW (Continuous Bag of Words)**: Predicts a target word based on context words[1].
        - **Skip-gram**: Uses a target word to predict surrounding words[1].
      - **FastText**: Extension of Word2Vec that considers the internal structure of words at the subword level[1].
      - **GloVe**: Captures global information about how words co-occur in the entire corpus[1].
      - **swivel**: Uses a co-occurrence matrix and is efficient for large datasets with distributed processing[1].

- **Document Embeddings**: Represent the meaning of larger text chunks (paragraphs or whole documents)[1].
  - **Bag of Words Models**:
    - **LSA (Latent Semantic Analysis)**: Uses a matrix of word counts and documents and applies dimensionality reduction techniques[1].
    - **LDA (Latent Dirichlet Allocation)**: Models each document as a mixture of topics[1].
  - **TF-IDF Based Models**: Weigh words based on their frequency in a document compared to the entire corpus; BM25 is a strong baseline[1].
  - **Doc2Vec**: Extends Word2Vec by adding a paragraph vector to the model[1].
  - **Deep Pre-trained Large Language Models**:
    - **BERT**: Uses the Transformer architecture and is trained on large datasets[1].
    - **Sentence-BERT, SimCSE, and E5**: Designed to produce good sentence embeddings[1].
    - **T5, PaLM, Gemini, GPT, and Llama**: Larger language models leading to better embedding models like GTR and SentenceT5[1].
    - **Matryx embeddings**: Allow you to choose the dimensionality of the embeddings[1].
    - **Multi-vector embeddings**: Deal with documents that contain both text and images (e.g., ColBERT, XTR, and Kpali)[1].
- **Image Embeddings**: Obtained by training convolutional neural networks (CNNs) or Vision Transformers on large image datasets[1].
- **Multimodal Embeddings**: Combine image embeddings with other types of embeddings (e.g., text embeddings) to create joint representations[1].
- **Structured Data Embeddings**: More application-specific due to the data's dependency on the schema and context[1].

        - **Techniques**: Dimensionality reduction techniques like PCA[1].
        - **User and Item Data**: Map users and items into the same embedding space for recommendation systems[1].
    - **Graph Embeddings**: Represent objects and their relationships within a network[1].
        - **Algorithms**: DeepWalk, Node2Vec, LINE, and GraphSage[1].

Training Embedding Models

- **Dual Encoder Architecture**: Uses an encoder for the query and an encoder for the documents/images[1].
- **Contrastive Loss**: Pulls embeddings of similar data points closer together while pushing dissimilar ones farther apart[1].
- **Training Stages**:
    - **Pre-training**: Training the model on a massive dataset to learn general representations[1].
    - **Fine-tuning**: Tuning the model on a smaller, task-specific dataset[1].
- **Fine-tuning Data Sets**: Created through manual labeling, synthetic data generation, model distillation, or hard negative mining[1].
- **Downstream Tasks**: Trained embeddings can be used for various tasks, such as classification[1].

Vector Search

- **Efficient Searching**: Finding items based on their meaning, not just keywords[1].
- **Process**: Compute embeddings for all data, store them in a vector database, embed the query into the same space, and find data items with the closest embeddings[1].
- **Approximate Nearest Neighbor (ANN) Search**: Used to speed up search in large datasets[1].
- **ANN Techniques**:

- **Locality Sensitive Hashing (LSH)**: Maps similar items to the same bucket using hash functions[1].
- **Tree-based Methods**: KD trees and ball trees partition the data space recursively[1].
- **Combining Hashing and Tree-based Approaches**: e.g., FAI with HNSW and Scan[1].
- **HNSW (Hierarchical Navigable Small World)**: Builds a hierarchical proximity graph[1].
- **Scan (Scalable Approximate Nearest Neighbor)**: Used in Google products and available through Vertex AI Vector Search[1].

Vector Databases

- **Specialized Systems**: Designed for high-dimensional data and similarity-based queries[1].
- **Hybrid Search**: Traditional databases are starting to add vector search capabilities[1].
- **Workflow**: Embed data, index vectors, embed query, and search for similar items[1].
- **Options**: Vertex Vector Search, AlloyDB, Cloud SQL for PostgreSQL, Pinecone, Weaviate, ChromaDB[1].
- **Operational Considerations**: Scalability, availability, consistency, updates, backups, and security[1].

Applications of Embeddings and Vector Stores

- **Applications**: Information retrieval, recommendation systems, semantic text similarity, classification, clustering, and reranking[1].
- **Combining with ANN Search**: Enables powerful applications like RAG, large-scale search engines, personalized recommendation systems, anomaly detection, and few-shot classification[1].
- **Ranking**: Embeddings are often used in the first stage of ranking for large-scale applications[1].
- **RAG**: Enhances language model accuracy[1].

- **Importance of Sources**: Providing sources for retrieved information to enhance user trust[1].