Job Description Duplicate Detection

Ji Qi

Jun 11, 2024

1 Part 1: Generating Embeddings

1.1 Objective

Generate embeddings for job descriptions to be used for duplicate detection.

1.2 Method

- Utilized Sentence Transformers (all-MiniLM-L6-v2) to generate embeddings.
- Cleaned job descriptions to remove HTML tags using BeautifulSoup.
- Converted job descriptions to embeddings and saved them in a NumPy array.

1.3 Rationale for Choosing Sentence Transformers

Sentence Transformers were chosen over other models like CBOW for several reasons:

- **Contextual Understanding**: Sentence Transformers capture the contextual meaning of words in a sentence, which is crucial for understanding the nuances in job descriptions. In contrast, CBOW averages the word embeddings, losing the context.
- **Pre-trained Models**: Sentence Transformers come with pre-trained models like all-MiniLM-L6-v2 that are optimized for sentence-level tasks, making them highly effective for generating meaningful embeddings quickly.
- **Performance**: Sentence Transformers generally provide better performance in terms of capturing semantic similarities and differences, which is essential for accurate duplicate detection.
- **Ease of Use**: The Sentence Transformers library is easy to integrate and use, with straightforward methods for encoding sentences into embeddings.

2 Part 2: Implementing Milvus for Duplicate Detection

2.1 Objective

Set up Milvus, insert embeddings, and search for potential duplicates.

2.2 Steps

1. **Set up Milvus**: Used Docker to set up a Milvus instance.
2. **Insert Embeddings**: Inserted embeddings into Milvus and created an index for efficient searching.
3. **Search for Duplicates**: Performed batch searches to identify potential duplicate job descriptions based on L2 distance metric.

2.3 Details

- **Connections**: Established a connection to the Milvus server using the ‘pymilvus‘ library. This connection allows communication between the Python application and the Milvus server.
- **Schemas**: Defined a schema for the collection to specify the data structure. The schema included fields for ‘job_id‘ (an auto-incrementing primary key) and ‘embedding‘ (a vector of floats representing the job description embedding).
- **Collections**: Created a collection named ‘job_postings_v5‘ in Milvus to store the embeddings. A collection in Milvus is analogous to a table in a relational database.
- **Inserting Embeddings**: Converted the embeddings to a list format and inserted them into the collection. Milvus automatically generated unique IDs for each embedding.
- **Indexing**: Created an index on the ‘embedding‘ field to improve search performance. The IVF_FLAT index type with the L2 distance metric was used.
- **Loading the Collection**: Loaded the collection into memory to make it ready for search operations.
- **Batch Search**: Performed batch searches by dividing the embeddings into smaller batches to manage memory usage and stay within Milvus's request limits. For each batch, searched the entire collection to find the top 2 most similar embeddings.
- **Thresholding**: Applied a similarity threshold to identify potential duplicates. Embeddings with a similarity score below the threshold were considered duplicates.

2.4 Determining the Threshold and Metrics Used

- **Empirical Analysis**: Started with a histogram of similarity scores in Figure 1 to understand the distribution and identify clusters or gaps. From the plot, we can see that 0.05 might be a good threshold.
- **Domain Knowledge**: Leveraged domain knowledge to set an initial threshold. For example, in certain domains, embeddings with a similarity score below 0.1 might be considered very similar or identical.
- **Metrics Used**: Used L2 (Euclidean) distance as the metric for similarity. Lower distances indicate higher similarity.

2.5 Results

By setting the threshold to be 0.05 and making sure that we don't compare the embedding with itself, we got 167408 duplicated pairs.

3 Part 3: Containerization

3.1 Objective

Encapsulate the environment, dependencies, and code into Docker containers and set up the application using Docker Compose.

3.2 Steps

1. **Create Dockerfile**: Defined the environment, dependencies, and entry point for the application.
2. **Set Up Docker Compose**: Configured services for Milvus and the application.
3. **Build and Run Containers**: Detailed instructions provided in the README for building and running the Docker containers.

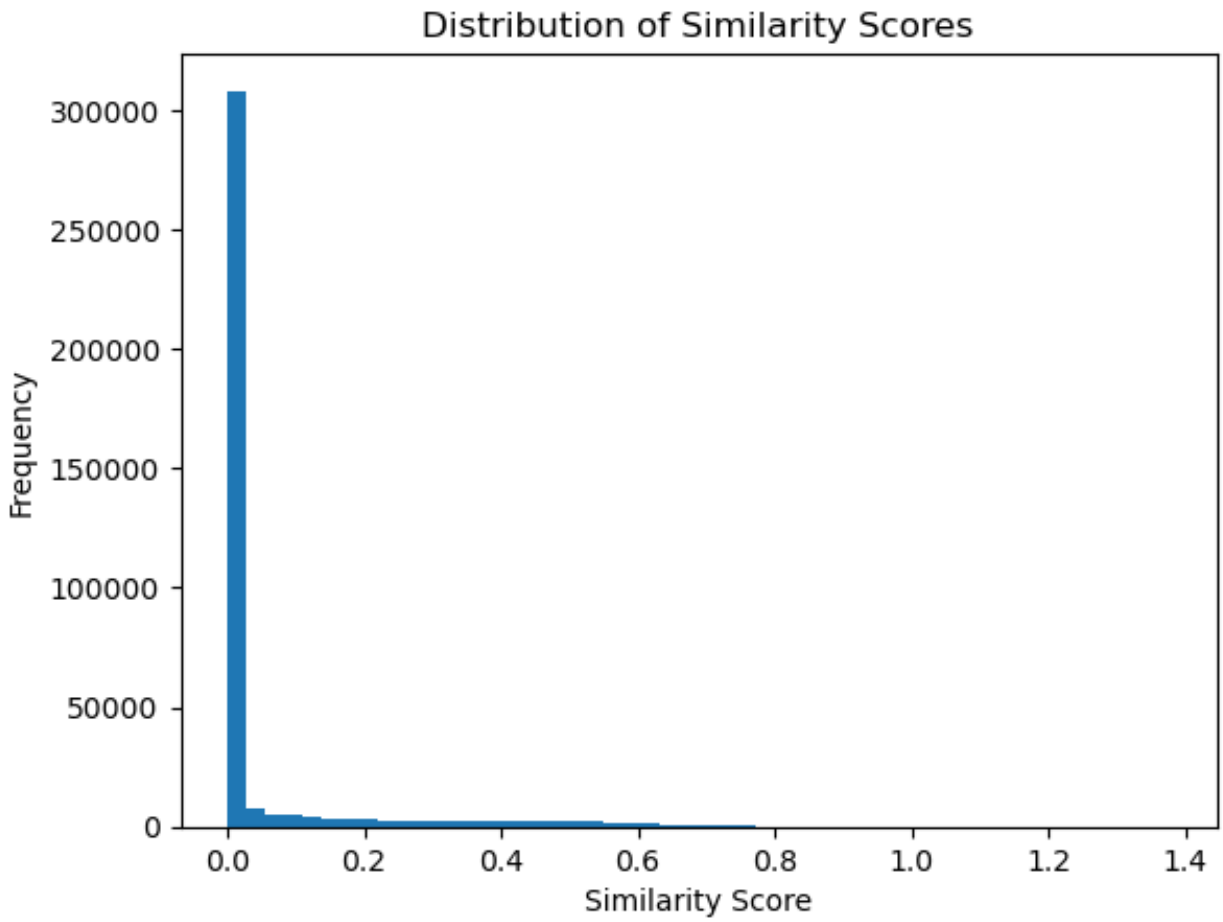

Figure 1: Distribution of Similarity Score

4 Other Use-Cases for Generated Embeddings

4.1 Beyond Duplicate Detection

The embeddings generated for job descriptions have several other potential use-cases:

- **Content-Based Recommendation Systems**: Recommend similar job postings to users based on the job descriptions they have viewed or applied to.
- **Semantic Search**: Enhance search functionality by finding job postings that are semantically similar to a user's search query, improving search relevance.
- **Clustering**: Cluster job postings into categories or topics based on their embeddings, aiding in the organization and navigation of job postings.
- **Visualization**: Visualize job postings in a lower-dimensional space (e.g., using t-SNE or PCA) to understand the relationships and distribution of job descriptions.

5 Conclusion

This project demonstrated the generation of embeddings from job descriptions, implementation of Milvus for duplicate detection, and containerization of the environment using Docker and Docker Compose. The solution effectively identifies duplicates and can be easily deployed and scaled. Additionally, the generated

embeddings have multiple use-cases beyond duplicate detection, offering versatile applications in recommendation systems, semantic search, clustering, anomaly detection, visualization, transfer learning, and text classification.