DataNeuron Semantic Textual Similarity Assessment Report

Submitted by: AMRESH KUMAR YADAV

Date: May 20, 2025

1 Introduction

This report outlines the core approach for the DataNeuron Semantic Textual Similarity (STS) assessment, covering Part A (model development) and Part B (API deployment). The task involves building an unsupervised model to quantify semantic similarity between pairs of text paragraphs, scored from 0 (highly dissimilar) to 1 (highly similar), and deploying it as a server API endpoint.

2 Part A: Semantic Textual Similarity Model

The STS model leverages an unsupervised approach due to the unlabeled dataset provided. I used the `sentence-transformers` library with the `all-MiniLM-L6-v2` model, a lightweight transformer optimized for semantic similarity tasks.

- **Embedding Generation**: The model encodes each text pair into dense vector representations (embeddings) using `all-MiniLM-L6-v2`, which is efficient for short to medium-length texts and performs well on STS benchmarks.
- **Preprocessing**: A custom `preprocess_text` function normalizes inputs by converting to lowercase, removing excessive whitespace, and stripping special characters (except `.`, `!`, `?`). This ensures robustness against noisy or inconsistent text.
- **Similarity Computation**: Cosine similarity is calculated between the normalized embeddings of the two texts, producing a score in the range [0, 1]. The score is clamped to ensure compliance with the task's requirements.

The unsupervised approach avoids the need for labeled training data, relying on the pre-trained model's ability to capture semantic relationships. The choice of `all-MiniLM-L6-v2` balances performance and computational efficiency, suitable for deployment in resource-constrained environments.

3 Part B: API Deployment

The model is exposed as a server API endpoint using FastAPI, containerized with Docker, and deployed on Google Cloud Run for scalability and ease of access.

- **FastAPI Implementation**: The API is built with FastAPI, providing a `/similarity` POST endpoint that accepts JSON input (`{"text1": "...", "text2": "..."}`) and returns a JSON response (`{"similarity score": float}`). Input validation ensures texts are non-empty and within the 10,000-character limit.
- **Error Handling**: Robust error handling includes HTTP exceptions for invalid inputs (422), model loading failures (503), and processing errors (500). A logging system with request IDs aids debugging.
- **Deployment**: The application is containerized using a `Dockerfile` based on `python:3.10-slim`. Dependencies are managed via `requirements.txt`, including `sentence-transformers`, `fastapi`, and `uvicorn`. The container is deployed to Google Cloud Run, with the endpoint accessible at `[Insert Live API Endpoint, e.g., https://dataneuron-project-366741234981.us-central1.`

A `/health` endpoint monitors the service's status, confirming model loading and API readiness. The deployment uses free-tier resources to comply with the task's recommendation.

4 Assumptions and Challenges

Assumptions:

- The dataset's lack of labels necessitates an unsupervised approach, leveraging pre-trained embeddings.
- The `all-MiniLM-L6-v2` model is assumed sufficient for the dataset's text complexity based on STS benchmark performance.
- The cloud provider (Google Cloud Run) is chosen for its free tier and scalability, aligning with the task's guidelines.

Challenges:

- **Model Loading Time**: Initial model loading can be slow due to downloading `all-MiniLM-L6-v2`. Retry logic (3 attempts, 2000ms wait) mitigates transient failures.
- **Text Preprocessing**: Handling diverse text formats required robust preprocessing to ensure consistent embeddings.
- **Deployment Debugging**: Ensuring the Docker container listens on the correct port (8080) and integrates with Cloud Run's environment variables required careful configuration.

5 Conclusion

The STS solution uses a proven unsupervised approach with `sentence-transformers` for Part A, delivering reliable similarity scores. Part B's FastAPI deployment on Google Cloud Run ensures a scalable, accessible endpoint with robust error handling. The implementation meets all specified requirements, including the exact request-response format, and is optimized for performance and maintainability.