**Part A: Semantic Similarity Algorithm**

The primary objective of Part A was to compute the semantic similarity between two text passages. To achieve this, a **hybrid multi-metric approach** was employed, combining **lexical, semantic, and sequence-based methods**:

1. **Text Preprocessing**
   * Each input text was cleaned to remove punctuation, excessive whitespace, and stopwords.
   * Text normalization involved converting all characters to lowercase.
   * Words were lemmatized using NLTK’s WordNet lemmatizer, accounting for their grammatical roles (nouns, verbs, adjectives, adverbs) to preserve meaning while reducing inflectional forms.
2. **Similarity Metrics**  
   The final similarity score was computed by aggregating three complementary metrics:
   * **Cross-Encoder Similarity**:  
     A transformer-based CrossEncoder (cross-encoder/stsb-roberta-base) was used to compute a direct similarity score for the text pair. CrossEncoders jointly encode the pair and predict a similarity value, capturing nuanced semantic relationships.
   * **SBERT Embedding Similarity**:  
     Sentence embeddings were generated using all-MiniLM-L6-v2 from Sentence Transformers. Cosine similarity between the embeddings provided a vector-based semantic similarity measure, suitable for capturing meaning even if words differ.
   * **Lexical & Sequence Similarity (Jaccard + SequenceMatcher)**:  
     A combined lexical approach was used to compute:
     + **Token Jaccard Similarity**: Measures overlap of unique words between texts.
     + **Sequence Matcher Ratio**: Captures sequence-level similarity, reflecting word order and phrase structure.  
       The average of these two metrics gave a robust lexical similarity score.
3. **Final Similarity Score**  
   The three metrics were averaged to compute a **final similarity score**, balancing semantic understanding and lexical overlap. This multi-faceted approach ensured accurate similarity estimation even in the presence of synonyms, paraphrasing, or partial overlaps.

**Part B: Deployment**

The primary goal of Part B is to deploy the endpoint in a server so that it can be publicly accessed. I have used Google Cloud Platform to deploy the NLP based algorithm. I have created a Flask based ‘/predict’ endpoint through which text1 and text2 can be taken as input as json format.

After filling up the csv file, I wrote a code in app.py which contains a route named ‘/predict’. At first it takes out the cross-encoding score, then the sbert similarity score and then the jaccard score, takes their mean and then outputs the similarity score in json format.  
  
The time taken for generating the similarity score is a bit more due to the large size of CrossEncoder (cross-encoder/stsb-roberta-base) and SBERT(all-MiniLM-L6-v2). For checking it locally, I have checked the endpoints in Postman API and it was working absolutely fine.  
  
Then for hosting, at first I tried to use Render. But due to the large size and RAM specification I shifted over to Google Cloud Platform and hosted the endpoints.  
  
Hosted endpoint : <http://35.199.145.92:8000/predict>  
  
I have also shared everything in GITHUB :  
Link : https://github.com/xEspix/DataNeuron-Assignment