ASSESSMENT FOR DATA SCIENCE

Part A: Cosine Similarity Rating

Introduction

Cosine Similarity is a metric used to measure how similar two sentence/text based columns are. It calculates the cosine of the angle between two vectors, representing the documents, in a multidimensional space. In the context of natural language processing (NLP), Cosine Similarity is often used to assess the similarity between text documents.

Additional : Can use the pretrained model Or LLM as well. eg : Hugging Face Transformer Library ,langchain etc.

Methods

Text Preprocessing:

- Lowercasing: Convert all text to lowercase to ensure uniformity.
- HTML Tag Removal: Remove HTML tags from text.
- Numerical Digit Removal: Remove numerical digits from text.
- Special Character Removal: Remove special characters from text.
- Punctuation Removal: Remove punctuation from text.
- Stopword Removal: Remove common English stopwords.
- Lemmatization: Reduce words to their base or root form.

Vectorization:

- Count Vectorization: Convert preprocessed text into numerical vectors using CountVectorizer.

Cosine Similarity Calculation:

- Cosine Similarity Score: Calculate the cosine similarity score between pairs of text documents using sklearn's `cosine_similarity` function.

Similarity Rating:

- Mean Similarity Score: Calculate the mean similarity score for each pair of text documents.
- Thresholding: Assign a binary similarity rating (0 or 1) based on a predefined threshold.

Reasons for Methods

Text Preprocessing:

- Lowercasing: Ensures uniformity in text comparison.
- HTML Tag Removal: Eliminates irrelevant HTML tags that don't contribute to the meaning.
- Numerical Digit Removal: Focuses on the textual content by removing numerical noise.
- Special Character Removal: Reduces noise and focuses on meaningful content.
- Punctuation Removal: Eliminates punctuation, which doesn't contribute significantly to text similarity.
- Stopword Removal: Removes common words that do not carry much meaning.
- Lemmatization: Reduces words to their base form, capturing the core meaning.

Vectorization:

- Count Vectorization: Represents text as numerical vectors, enabling mathematical comparison.

Cosine Similarity Calculation:

- Cosine Similarity Score: Measures the cosine of the angle between vectors, providing a robust similarity metric.

Similarity Rating:

- Mean Similarity Score: Considers the overall similarity between pairs of text.
- Thresholding: Converts continuous similarity scores into binary ratings for practical use.

Future Work :

Custom Model : After getting the similarity rating we can use this as labels to train our custom model. In Custom model we use to train that model with an extinct dataset. For custom models using the “Binary Classification” model. For the tokenization method use DistilbertTokenizer.

Part B: Flask API

Introduction

A Flask API is created to expose the functionality of calculating cosine similarity ratings for pairs of sentences. The API receives POST requests with two sentences, preprocesses the input, calculates cosine similarity, and returns a similarity rating.

Note : I don't have any free AWS account to develop or any free server (I don't have much time to explore the new platform). So I have used the Flask method to host in the local system.

Methods

Flask Setup:

- Flask Instance: Create a Flask web application instance.

Text Preprocessing:

- Preprocess Function: Perform text preprocessing on input sentences.

Vectorization and Similarity Calculation:

- Vectorization: Use precomputed vectors from a sample DataFrame for Count Vectorization.
- Cosine Similarity Calculation: Utilize precomputed cosine similarity scores.

API Endpoint:

- Endpoint `/get_similarity`: Exposes the similarity calculation functionality via a POST request.

Response:

- JSON Response: Returns the similarity rating in JSON format.

Reasons for Methods

Flask Setup:

- Web Framework: Flask is a lightweight web framework suitable for small to medium-sized applications.
- RESTful API: Utilizes RESTful principles for simplicity and ease of use.

Text Preprocessing:

- Functionality Reusability: Centralizes text preprocessing logic for code reusability.
- Maintainability: Easier to update and modify preprocessing steps in one place.

Vectorization and Similarity Calculation:

- Efficiency: Precomputing vectors and similarity scores avoids redundant computations.
- Sample Data: Uses a sample DataFrame for demonstration, allowing easy integration with real data.

API Endpoint:

- POST Request Handling: Accepts POST requests to keep the API stateless.
- Structured Endpoint (`/get_similarity`): Follows a structured naming convention for clarity and consistency.

Response:

- JSON Format: Common and easy-to-parse format for API responses.
- Key-Value Pair: Provides a clear structure for conveying similarity ratings.

Conclusion

The combination of Cosine Similarity for text comparison and a Flask API for exposing the functionality in a user-friendly manner results in a practical and efficient system for calculating and obtaining similarity ratings between pairs of sentences. The methods employed ensure accuracy, reusability, and maintainability in both the similarity rating calculation and the API design.

Submitted By:
Kuldeep Limbachiya