Similarity Module Function

# Overview

The scoring module, now implemented as a Flask microservice, evaluates the compatibility between resumes and job descriptions. It leverages advanced text analysis techniques, including natural language processing (NLP), embeddings, and cosine similarity measures.

# Scoring Module Functions

## 1. calculate\_cosine\_similarity

* Purpose:
  + The function computes the cosine similarity between two lists of strings. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size.
* Inputs: Two lists (list1, list2).
* Process:
  + The function first converts both lists of strings into a single string by joining each list's elements with spaces. This is akin to creating two 'documents' from the input lists.
  + It then employs TfidfVectorizer from the Scikit-learn library to transform these concatenated strings into TF-IDF (Term Frequency-Inverse Document Frequency) vectors. TF-IDF is a numerical statistic that reflects how important a word is to a document in a collection or corpus.
  + Once the TF-IDF vectors are obtained, the function calculates the cosine similarity between these two vectors. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.
  + The result is a matrix where each element represents the cosine similarity between the corresponding elements in the two vectors.
* Output:
  + A single floating-point number representing the cosine similarity between the two lists of strings.
  + The output is in the range of 0 to 1, where 1 means perfectly similar and 0 means completely dissimilar.
* Error Handling:
  + The function includes a try-except block to handle any exceptions that may arise during the computation process, such as errors in vectorization or similarity calculation.
  + If an exception occurs, it logs the error message and returns a similarity score of 0, indicating no similarity.
* Usage: Ideal for comparing skill lists or textual attributes.

## 2. get\_embedding

* Purpose:
  + This function generates a numerical representation, or embedding, of a given text string using a pre-trained BERT model. Embeddings capture the contextual nuances of the text, making them useful for various natural language processing tasks.
* Inputs: A text string (text).
* Process:
  + The function utilizes a tokenizer and model from the Hugging Face Transformers library, specifically the "bert-base-uncased" model.
  + The input text is first tokenized using the BERT tokenizer. This process converts the text into a format suitable for input into the BERT model, including options like padding and truncation for consistent input length.
  + The tokenized input is then fed into the BERT model to obtain the embeddings. The BERT model returns a series of hidden states, representing the text in a high-dimensional space.
  + The function computes the mean across all tokens' hidden states to get a single vector representation of the input text.
  + This vector is detached from the PyTorch computation graph and converted to a NumPy array for easy handling in downstream tasks.
* Output:
  + A NumPy array representing the text's embedding. This array is a fixed-size vector, where each element represents a feature learned by the model about the input text.
  + Returns None if an error occurs during the embedding process.
* Error Handling
  + Includes a try-except block to catch and log any exceptions that occur during tokenization or model inference.
  + If an error occurs, the function logs the error along with the input text that caused it and returns None.
* Usage: Converts textual data into numerical form for similarity calculations.

## 3. calculate\_similarity

* Purpose: Computes similarity between two texts based on embeddings.
* Inputs: Two text strings (text1, text2).
* Process: Obtains embeddings for each text and calculates cosine similarity.
* Output: Similarity score based on text embeddings.
* Error Handling: Logs errors during processing.
* Usage: Useful for text-based comparisons, like job responsibilities.

## 4. median\_similarity

* Purpose:
  + This function computes the median similarity score between elements of two lists. It is designed to evaluate the overall similarity between groups of items, where each item is compared pairwise across both lists.
* Inputs:
  + list1: The first list containing elements to be compared. These elements could be text strings, numerical data, or any other format compatible with the calculate\_similarity function.
  + list2: The second list, similar in nature to list1, against which the elements of list1 are compared.
* Process:
  + The function iterates over the Cartesian product of list1 and list2, effectively comparing each element of list1 with each element of list2 in pairs.
  + For each pair, it calculates the similarity score using a calculate\_similarity function. This function is assumed to return a similarity score or None if the similarity cannot be calculated.
  + All valid similarity scores (non-None values) are collected in a list.
  + After all pairwise comparisons, the function calculates the median of these scores.
  + The median is used as it is a robust statistical measure that is less affected by outliers in the similarity scores, providing a more balanced view of the overall similarity.
* Output:
  + A single numeric value representing the median of all pairwise similarity scores. This value gives a central tendency measure of the similarity between the two lists.
  + If no valid similarity scores are found, the function returns 0.
* Error Handling:
  + The function checks for empty input lists at the beginning and logs a warning if either list is empty, returning 0 as the median similarity.
  + It also handles cases where no valid similarity scores could be computed (all pairs yield None), logging a warning and returning 0.
* Usage
  + Useful in scenarios requiring a holistic similarity assessment between two sets of elements.

## 5. location\_similarity\_median

* Purpose:
  + Calculates a similarity score based on the geographical distance between locations provided in two lists. The function is tailored to assess how geographically close or distant the sets of locations are from each other, returning a similarity score where a higher value indicates closer proximity.
* Inputs: Two lists of location names (list1, list2).
* Process
  + The function uses Nominatim from the Geopy library to geocode the locations into latitude and longitude coordinates.
  + It iterates over every possible pair of locations, one from each list, and calculates the geographical distance between each pair using the geopy.distance function.
  + The geographical distances are collected in a list.
  + If valid distances are found, the function calculates their median. This median distance is then normalized against the maximum possible distance on Earth (approximated as 20,000 kilometers) to derive a similarity score. The score is calculated as 1 - (median\_distance / max\_distance), ensuring that closer locations result in a higher similarity score.
  + The function includes a sleep call to throttle the requests to the geocoding service, preventing potential rate limit issues.
* Output:
  + A similarity score between 0 and 1, where 1 indicates very close proximity (geographically similar) and 0 indicates maximum possible distance.
  + If no valid location pairs are found or geocoding fails for all pairs, the function returns 0.
* Error Handling
  + The function checks for empty input lists and logs a warning if either is empty.
  + Handles geocoding failures and exceptions, logging warnings and errors as they occur, and continues processing other pairs.
  + If no valid distances are calculated, logs a warning indicating no valid location pairs were found.
* Usage: Evaluates geographical suitability for job matching.

## 6. industry\_similarity\_score

* Purpose:
  + Calculates the semantic similarity between two text strings, specifically tailored for industry or domain-related texts. This is useful in contexts where understanding the closeness or relevance of two industry-specific descriptions is needed, such as comparing a job description with a resume.
* Inputs:
  + text1: A string containing text related to an industry, such as a segment from a resume.
  + text2: Another string containing industry-related text, for instance, from a job description.
* Process:
  + The function utilizes the SentenceTransformer model, specifically 'all-MiniLM-L6-v2', to generate embeddings for each input text. These embeddings are numerical representations that capture the semantic essence of the texts.
  + It then computes the cosine similarity between these embeddings using the util.pytorch\_cos\_sim function. Cosine similarity measures the cosine of the angle between two vectors, providing a similarity score between -1 and 1.
  + The function operates in a way that higher similarity scores indicate more significant semantic relatedness between the texts.
* Output:
  + A single floating-point number representing the similarity score. This score ranges from -1 (completely dissimilar) to 1 (identical), with higher scores indicating greater similarity between the two input texts.
* Error Handling:
  + The function is wrapped in a try-except block to catch and log any exceptions that occur during the process of encoding the texts or calculating similarity.
  + In case of an exception, it logs the error details and returns a default similarity score of 0, implying no similarity.
* Usage:
  + This function can be employed in scenarios like analyzing the similarity between a candidate's resume and a specific job description in the context of recruitment, or comparing textual content across different industry documents to find semantic resemblances.

# Improvements

* Enhance Text Preprocessing: Incorporate more robust text preprocessing to improve the quality of input data for each function. This might include advanced techniques for handling domain-specific terminology or jargon.
* Refinement of Embedding Techniques: We will explore domain-specific embedding models better suited to capturing technical and specialized language nuances. This will enable a more precise understanding and representation of industry-specific terminologies and concepts.
* Enhanced Geocoding Accuracy: We aim to upgrade our location similarity assessments by integrating more detailed geocoding capabilities. This enhancement will allow for a finer regional understanding and more accurate geographic matching.
* Semantic Analysis Refinement: For the industry/domain similarity aspect, we are considering fine-tuning our SentenceTransformer model with a corpus tailored more specifically to our domain requirements. This refinement will improve our system's ability to discern and align with the semantic intricacies of different industry sectors.