NLTK (Accuracy: 88.4119%)

The provided code utilizes TensorFlow, TensorFlow Hub, and NLTK to calculate the average cosine similarity between sentences in two text files using the Universal Sentence Encoder (USE). Here's an analysis of the code:

1. **Text Preprocessing:** The `preprocess_text` function tokenizes the input text, removes stopwords, punctuation, and lemmatizes words. This preprocessing step helps in cleaning and standardizing the text data before embedding.

2. **Universal Sentence Encoder (USE):** The code loads the Universal Sentence Encoder (USE) model from TensorFlow Hub (`https://tfhub.dev/google/universal-sentence-encoder/4`). USE is a pre-trained model that encodes sentences into fixed-dimensional vectors, making it suitable for calculating sentence similarity.

3. **Loading Text Files**: It reads and loads the content of two text files (`data1.txt` and `data2.txt`) into separate lists, considering the first 10,000 lines from each file. The code assumes that these files contain text data.

4. **Text Preprocessing (Continued):** The lines from both files undergo the same preprocessing steps to ensure consistent embeddings.

5. **Cosine Similarity Calculation:** For each pair of corresponding lines from the two files, the code calculates cosine similarity using the USE embeddings. The `np.inner` function computes the cosine similarity between the embeddings of two sentences.

6. **Average Similarity:** The code accumulates the cosine similarity scores for all sentence pairs and calculates the average similarity over all pairs.

7. **Output**: Finally, it prints the average cosine similarity score, representing the semantic similarity between sentences in the two files based on their embeddings.

This code provides an efficient way to calculate sentence similarity using the Universal Sentence Encoder and is suitable for various natural language processing tasks that involve text comparison or similarity analysis.

Conceptnet (Accuracy: 68.6315%)

1**. Efficient Semantic Similarity Calculation**: The code efficiently calculates semantic similarity between sentences using ConceptNet. By dividing the task into smaller chunks and processing them concurrently, it significantly improves execution speed, making it suitable for large datasets.

2. **API Request Handling:** The code makes use of the `requests` library to interact with the ConceptNet API. It handles potential API errors by checking the HTTP status code and gracefully handling situations where the API request fails.

3**. Preprocessing for Text Comparison**: Text preprocessing is essential for accurate semantic similarity calculations. The code effectively preprocesses the text data, removing punctuation and converting text to lowercase to ensure consistent and meaningful comparisons.

4. **Custom Range Specification:** The code allows you to specify a custom range of lines to calculate semantic similarity for, providing flexibility in analyzing specific sections of the text files. This feature is useful for fine-grained analysis.

5. **Average Similarity Score**: After processing all sentence pairs, the code calculates the average similarity score, providing a summary measure of semantic similarity between the two sets of sentences.

6. **Parallel Processing:** The use of a ThreadPoolExecutor for parallel processing is a key optimization. It takes advantage of multiple CPU cores to perform calculations concurrently, resulting in significant time savings for large datasets.

7. **Configurability**: The code is configurable, allowing you to adjust parameters such as the number of concurrent threads (`num_threads`) and the range of lines to process. This flexibility enables optimization based on your system's capabilities and specific analysis requirements.

Gensim (Accuracy: 98.217163%)

This code snippet demonstrates how to calculate semantic similarity between sentences using a pre-trained Word2Vec model from Gensim. Here's an analysis of the code:

1. **Word2Vec Model Loading**: The code loads a pre-trained Word2Vec model using Gensim's `gensim.downloader` module. In this case, the model used is the 'word2vec-google-news-300', which contains word embeddings for a vast vocabulary.

2. **Text Preprocessing:** The `preprocess_text` function is defined to preprocess input text by converting it to lowercase and removing punctuation. This step helps ensure consistent embeddings for sentence comparison.

3. **Cosine Similarity Calculation**: The `calculate_cosine_similarity` function calculates the cosine similarity between two sentences using Word2Vec embeddings. It tokenizes the sentences, filters out tokens not present in the Word2Vec model's vocabulary, and calculates the mean vector for each sentence. Then, it computes the cosine similarity between the two mean vectors.

4. **Accuracy Calculation:** The `calculate_accuracy` function calculates the average semantic similarity between pairs of sentences from two text files ('data1.txt' and 'data2.txt'). It iterates through a range of lines (from line 2 to line 999) and computes the cosine similarity for each pair. The average similarity score is then calculated and returned.

5. **File Reading:** The code reads the lines from both text files ('data1.txt' and 'data2.txt') and stores them in `file1_lines` and `file2_lines`, respectively.

6. **Calculation and Output**: The code calculates the semantic similarity accuracy using the specified range of lines and the pre-trained Word2Vec model. It prints the average similarity score as the result.

This code provides a straightforward way to measure the semantic similarity between sentences using Word2Vec embeddings and is applicable to various natural language processing tasks that require text comparison or similarity analysis.

Sentence Transformer (BERT) (Accuracy: 97.1153%)

This code snippet demonstrates how to calculate the average cosine similarity between sentences using a pre-trained BERT-based model from the `sentence_transformers` library. Here's an analysis of the code:

1. **Text Preprocessing**: The `preprocess_text` function tokenizes the input text into words, removes stopwords, punctuation, and lemmatizes the words. This preprocessing step helps ensure that the input text is clean and standardized before generating sentence embeddings.

2. **Pretrained BERT-Based Model**: The code loads a pre-trained BERT-based model named "bert-base-nli-stsb-mean-tokens" using the `SentenceTransformer` class. This model is fine-tuned for semantic similarity tasks and provides sentence embeddings.

3**. File Loading**: The code reads and loads the content of two text files ('data1.txt' and 'data2.txt') into separate lists, considering lines 2 to 599 from each file. These lines are read and used for similarity calculation.

4. **Cosine Similarity Calculation**: For each pair of corresponding lines from the two files, the code calculates cosine similarity using BERT embeddings. It encodes each line into a tensor using the BERT model and computes the cosine similarity using the `util.pytorch_cos_sim` function from the `sentence_transformers` library.

5. **Average Similarity:** The code accumulates the cosine similarity scores for all sentence pairs and calculates the average similarity over all pairs.

6. **Output:** Finally, it prints the average cosine similarity score, representing the semantic similarity between sentences in the two files based on their BERT embeddings.

This code provides an efficient way to calculate sentence similarity using a BERT-based model, making it suitable for various natural language processing tasks that involve text comparison or similarity analysis.