torch.nn.functional.cosine\_similarity

1. Library: PyTorch

2. Use Case: Primarily designed for deep learning applications, where data is often in the form of tensors.

3. Input: Expects PyTorch tensors as input. It computes the cosine similarity between tensors along a specified dimension.

4. Output: Returns a tensor containing the cosine similarity scores.

5. Flexibility: Offers more control over which dimensions to compute the similarity on, making it suitable for batch operations in deep learning models.

6. Example Usage:

Normalization and Similarity Calculation: If you plan to use these vectors for similarity calculations (e.g., cosine similarity), ensure that the vectors are normalized if your downstream application requires it. TF-IDF vectors are typically normalized by the TfidfVectorizer, but combining them with BoW vectors (which are not normalized by default) means you might need to apply normalization to the combined vectors manually.

For cosine similarity calculations, especially in the context of text processing with methods like TF-IDF or Bag of Words (BoW), L2 normalization (also known as Euclidean normalization) is the most suitable and commonly used normalization technique. L2 normalization adjusts the vectors in such a way that they have a unit length (Euclidean norm) of 1. This normalization is crucial for cosine similarity because it ensures that the similarity measure focuses on the direction of the vectors rather than their magnitude, which aligns with the definition and intention of cosine similarity.

Key Differences

1. Interface and Input Types: PyTorch's function is designed for tensors and deep learning models, while scikit-learn's is geared towards traditional ML models with NumPy arrays or sparse matrices.

2. Dimensionality Control: PyTorch allows specifying the dimension along which to compute similarity, which is useful for batch operations in neural networks. Scikit-learn computes similarity across the feature dimension by default, suited for pairwise comparisons in ML tasks.

3. Both functions calculate the cosine similarity, which measures the cosine of the angle between two vectors, indicating how similar they are in orientation in the vector space. The choice between them depends on the specific requirements of your project, including whether you're working within a deep learning context (PyTorch) or a more traditional machine learning setup (scikit-learn).

sklearn.metrics.pairwise.cosine\_similarity

1. Library: scikit-learn

2. Use Case: Designed for machine learning applications, particularly with data in matrix form.

3. Input: Expects NumPy arrays or sparse matrices as input. It can compute the similarity between all pairs in two sets of vectors or within a single set.

4. Output: Returns a NumPy array containing the cosine similarity scores between all pairs of vectors.

5. Flexibility: Primarily used for pairwise similarity calculations, without explicit control over the dimension (assumes the last dimension represents features).

Example Usage:

When comparing texts using TF-IDF (Term Frequency-Inverse Document Frequency) or BoW (Bag of Words) for methods such as cosine similarity, it's crucial to process all texts within the same feature space. This means that each text is transformed into a vector where each dimension corresponds to a specific term in the overall vocabulary of the dataset.