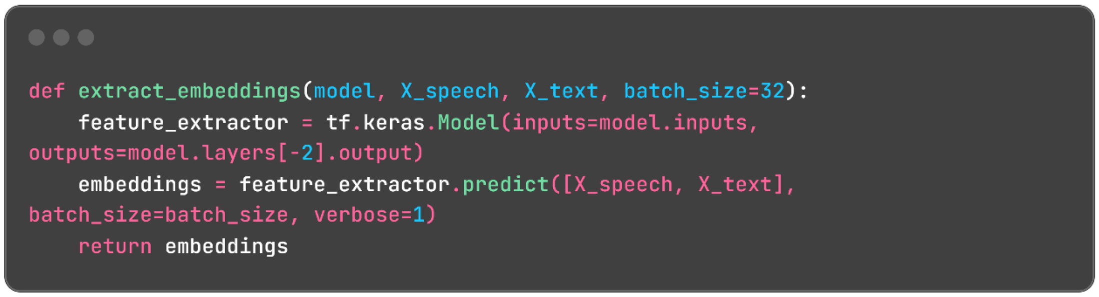
Construction of Class-wise Embeddings and Emotion Similarity Matrix in Multimodal Emotion Recognition

Overview

In this project, a deep learning model is trained to recognize emotions from both speech and text data. Beyond simple classification, we implement an emotion-aware loss function, which utilizes a similarity matrix representing semantic closeness between emotion classes. This document explains in detail how the final embeddings are extracted and how the emotion similarity matrix is constructed step-by-step.

Step 1: Extracting Embeddings from the Model

Once the model is trained, we want to capture the representation ("embedding") it has learned for each input sample before it outputs the final prediction. This representation captures the fused information from both speech and text modalities.

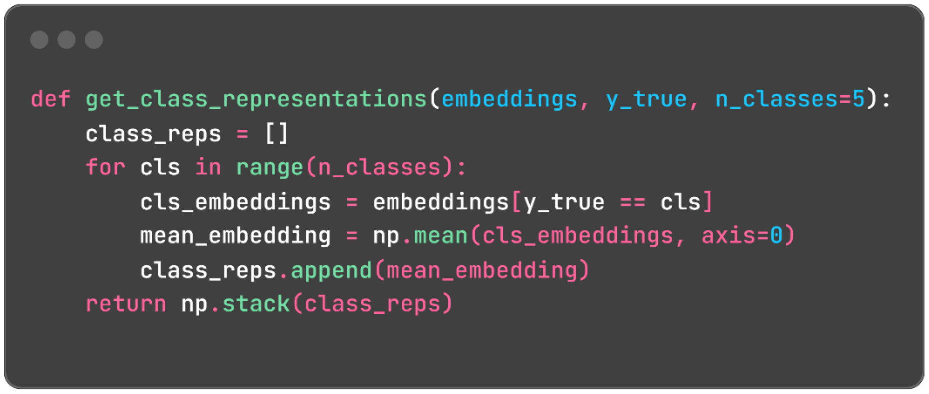


```
def extract_embeddings(model, X_speech, X_text, batch_size=32):
    feature_extractor = tf.keras.Model(inputs=model.inputs,
outputs=model.layers[-2].output)
    embeddings = feature_extractor.predict([X_speech, X_text],
batch_size=batch_size, verbose=1)
    return embeddings
```

- `model.layers[-2].output` refers to the last dense layer before the softmax output.
- `feature_extractor` becomes a new model that outputs the embedding vector for each input.
- For N test samples, we get N embedding vectors.

Step 2: Building Class-wise Representative Vectors

Each emotion class (e.g., happy, sad, angry, etc.) may contain multiple samples. We compute the mean embedding vector of all samples that belong to each class. These are known as **class representatives**.

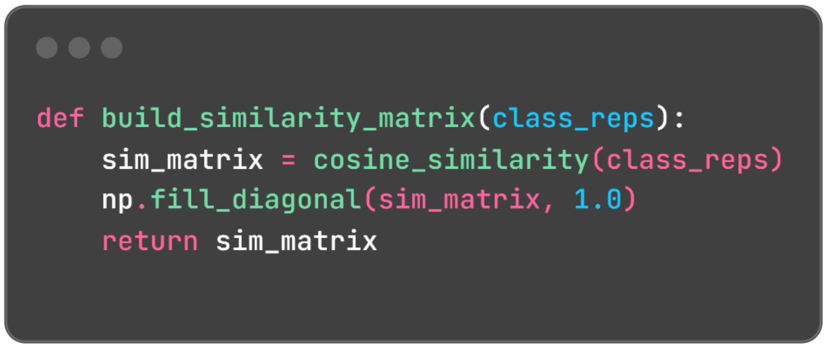


```python
def get_class_representations(embeddings, y_true, n_classes=5):
    class_reps = []
    for cls in range(n_classes):
        cls_embeddings = embeddings[y_true == cls]
        mean_embedding = np.mean(cls_embeddings, axis=0)
        class_reps.append(mean_embedding)
    return np.stack(class_reps)
```

- `cls_embeddings`: All embeddings whose label is `cls`.
- `mean_embedding`: Averaged vector across those samples.
- Final output: A matrix of shape `(n_classes, embedding_dim)`.

Step 3: Constructing the Similarity Matrix

Now that we have a representative vector for each class, we measure the similarity between them using cosine similarity.



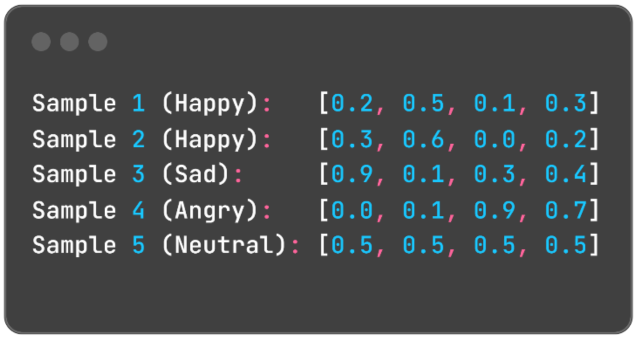
```python
def build_similarity_matrix(class_reps):
    sim_matrix = cosine_similarity(class_reps)
    np.fill_diagonal(sim_matrix, 1.0)
    return sim_matrix
```

- `cosine_similarity(A, B)` measures angular similarity between vectors A and B.
- `np.fill_diagonal(..., 1.0)` ensures each class has a similarity of 1.0 with itself.
- Final result: A symmetric matrix of shape `(n_classes, n_classes)`.

Example

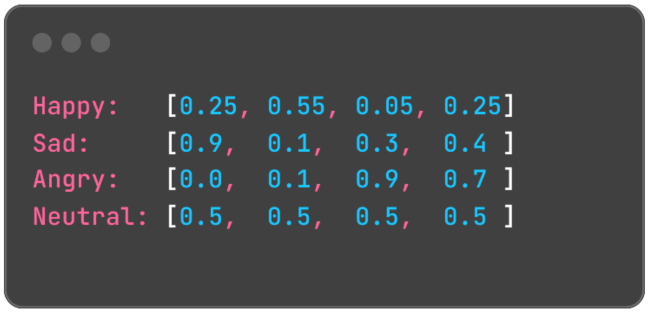
Assume we have 4 emotion classes labeled as: 0 (Happy), 1 (Sad), 2 (Angry), 3 (Neutral).

Sample output from the feature extractor (simplified, 4-dimensional embeddings):



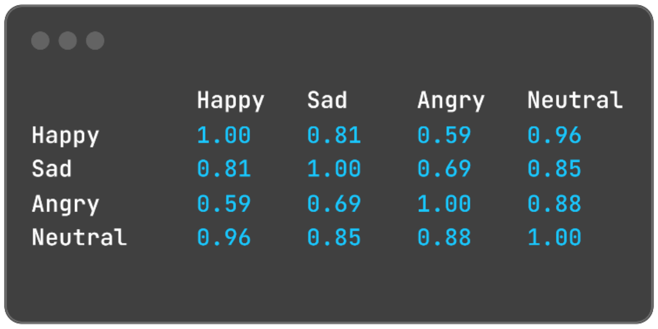
```
Sample 1 (Happy):   [0.2, 0.5, 0.1, 0.3]
Sample 2 (Happy):   [0.3, 0.6, 0.0, 0.2]
Sample 3 (Sad):     [0.9, 0.1, 0.3, 0.4]
Sample 4 (Angry):   [0.0, 0.1, 0.9, 0.7]
Sample 5 (Neutral): [0.5, 0.5, 0.5, 0.5]
```

Mean embeddings per class (computed from samples of each class):



```
Happy:   [0.25, 0.55, 0.05, 0.25]
Sad:     [0.9,  0.1,  0.3,  0.4 ]
Angry:   [0.0,  0.1,  0.9,  0.7 ]
Neutral: [0.5,  0.5,  0.5,  0.5 ]
```

Cosine Similarity Matrix (rounded):

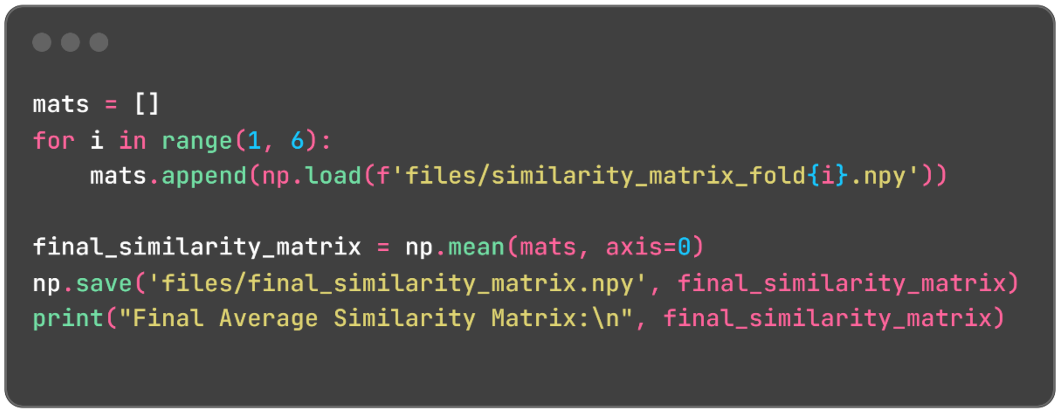


```
             Happy    Sad      Angry    Neutral
Happy        1.00     0.81     0.59     0.96
Sad          0.81     1.00     0.69     0.85
Angry        0.59     0.69     1.00     0.88
Neutral      0.96     0.85     0.88     1.00
```

This matrix is then used in the loss function to adjust error penalties based on emotion similarity.

Aggregating Fold-wise Matrices

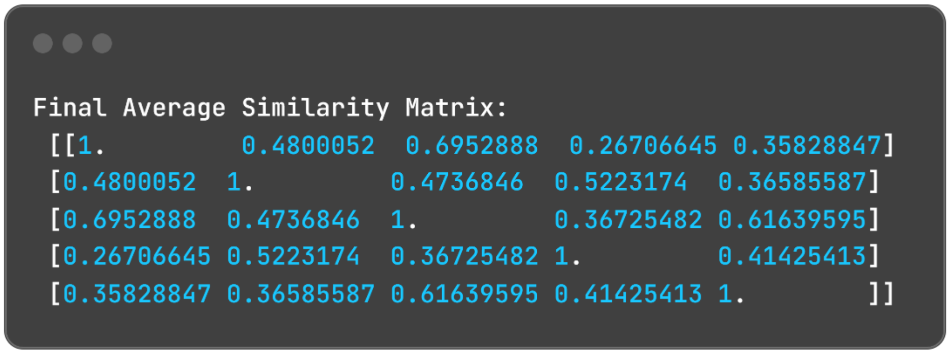
After training across 5 cross-validation folds, a similarity matrix is created for each fold based on test set embeddings. These 5 matrices are averaged to form a final, more stable similarity matrix:



```python
mats = []
for i in range(1, 6):
    mats.append(np.load(f'files/similarity_matrix_fold{i}.npy'))

final_similarity_matrix = np.mean(mats, axis=0)
np.save('files/final_similarity_matrix.npy', final_similarity_matrix)
print("Final Average Similarity Matrix:\n", final_similarity_matrix)
```

This averaged matrix reflects a more generalizable view of class similarities across different data partitions.



```
Final Average Similarity Matrix:
 [[1.         0.4800052  0.6952888  0.26706645 0.35828847]
 [0.4800052  1.         0.4736846  0.5223174  0.36585587]
 [0.6952888  0.4736846  1.         0.36725482 0.61639595]
 [0.26706645 0.5223174  0.36725482 1.         0.41425413]
 [0.35828847 0.36585587 0.61639595 0.41425413 1.        ]]
```

Purpose

This similarity matrix is crucial for building the **emotion-aware loss function**. It ensures that misclassifying two similar emotions (e.g., Happy vs. Neutral) is penalized less than confusing two dissimilar emotions (e.g., Happy vs. Angry).

Important Note

In the current implementation:

- During training, a **manual similarity matrix** is used as a placeholder.
- After training each fold, a **real similarity matrix** is built from embeddings.
- The final similarity matrix is obtained by averaging across folds and can be reused for fine-tuning or evaluation.