Notebook

November 29, 2024

Modular Code Breakdown

[ ]:
```

```

diarization_project/

main.py

config.py

audio_processing/

init.py

transcription.py

diarization.py

utils.py

evaluation/

init.py

metrics.py

clean_text.py

requirements.txt

config.py

This file contains all configuration variables like the Hugging Face token and model paths.

[ ]:
```
AUTH_TOKEN = "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"
OUTPUT_CSV = "/content/drive/MyDrive/output.csv"
INPUT_AUDIO = "/content/drive/MyDrive/Test/3.mp3"
```

audio_processing/transcription.py

Handles transcription using Whisper.

[ ]:
```
import whisper

def transcribe_audio(audio_file, model_name="tiny.en"):
    model = whisper.load_model(model_name)
    asr_result = model.transcribe(audio_file)
    return asr_result
```

audio_processing/diarization.py

Handles speaker diarization using Pyannote.

```
[ ]: from pyannote.audio import Pipeline

def diarize_audio(audio_file, auth_token):
    pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization-3.1",␣
 ↪use_auth_token=auth_token)
    diarization_result = pipeline(audio_file)
    return diarization_result
```

audio_processing/utils.py

Helper functions for processing transcription and diarization results.

```
[ ]: from pyannote.core import Segment

def get_text_with_timestamp(transcribe_res):
    return [(Segment(item['start'], item['end']), item['text']) for item in␣
 ↪transcribe_res['segments']]

def add_speaker_info_to_text(timestamp_texts, diarization_result):
    spk_text = []
    for seg, text in timestamp_texts:
        spk = diarization_result.crop(seg).argmax()
        spk_text.append((seg, spk, text))
    return spk_text

def merge_cache(text_cache):
    sentence = ''.join([item[-1] for item in text_cache])
    spk = text_cache[0][1]
    start = text_cache[0][0].start
    end = text_cache[-1][0].end
    return Segment(start, end), spk, sentence

def merge_sentence(spk_text):
    PUNC_SENT_END = ['.', '?', '!']
    merged_spk_text, pre_spk, text_cache = [], None, []
    for seg, spk, text in spk_text:
        if spk != pre_spk and pre_spk is not None and text_cache:
            merged_spk_text.append(merge_cache(text_cache))
            text_cache = [(seg, spk, text)]
        elif text and text[-1] in PUNC_SENT_END:
            text_cache.append((seg, spk, text))
            merged_spk_text.append(merge_cache(text_cache))
            text_cache = []
        else:
            text_cache.append((seg, spk, text))
        pre_spk = spk
    if text_cache:
        merged_spk_text.append(merge_cache(text_cache))
    return merged_spk_text
```

evaluation/metrics.py

Handles evaluation metrics like WER and ROUGE.

```
[ ]: from jiwer import wer
     from rouge_score import rouge_scorer

     def calculate_wer(reference, hypothesis):
         return wer(reference, hypothesis)

     def calculate_rouge(reference, hypothesis):
         scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'],␣
       ↪use_stemmer=True)
         return scorer.score(reference, hypothesis)
```

evaluation/clean_text.py

Handles text cleaning.

```
[ ]: import string

     def clean_text(text):
         translator = str.maketrans('', '', string.punctuation)
         return text.translate(translator).lower()
```

main.py

The entry point of the project.

```
[ ]: from config import AUTH_TOKEN, INPUT_AUDIO, OUTPUT_CSV
     from audio_processing.transcription import transcribe_audio
     from audio_processing.diarization import diarize_audio
     from audio_processing.utils import get_text_with_timestamp,␣
       ↪add_speaker_info_to_text, merge_sentence
     from evaluation.metrics import calculate_wer, calculate_rouge

     def main():
         # Step 1: Transcription
         asr_result = transcribe_audio(INPUT_AUDIO)

         # Step 2: Diarization
         diarization_result = diarize_audio(INPUT_AUDIO, AUTH_TOKEN)
```

```python
    # Step 3: Merge results
    timestamp_texts = get_text_with_timestamp(asr_result)
    spk_text = add_speaker_info_to_text(timestamp_texts, diarization_result)
    merged_text = merge_sentence(spk_text)

    # Output results
    print("Merged Transcription and Diarization:")
    for seg, spk, sent in merged_text:
        print(f"Speaker {spk}: {sent}")

    # Optional: Save to CSV
    with open(OUTPUT_CSV, 'w') as f:
        for seg, spk, sent in merged_text:
            f.write(f"{seg.start},{seg.end},{spk},{sent}\n")

if __name__ == "__main__":
    main()
```