Building and fine-tuning a speech-to-text (STT) model using the wav2vec 2.0 framework for Arabic language recognition.

Student: Salma Ahmed, Mena Majidi, Malak Reda

Standard Arabic Dataset:

The dataset used for Arabic speech recognition was sourced from the Mozilla Common Voice 11.0 dataset for Arabic (`mozilla-foundation/common_voice_11_0`, language code: 'ar'). The dataset consists of speech samples with corresponding transcripts in Arabic. The collected data was split into training+validation and test datasets as follows:

- Training and Validation Set: 'train+validation' split.
- Test Set: 'test' split.

The following columns were removed from both training and test datasets: `accent`, `age`, `client_id`, `down_votes`, `gender`, `locale`, `segment`, `up_votes`.

Preprocessing Steps

The preprocessing steps were as follows:

- Removal of special characters (e.g., punctuation, symbols) using regular expressions.
- Substitution of Arabic characters and noise removal (e.g., tashdid, tanwin).
- Mapping of sentences to input and label sequences suitable for Wav2Vec2 model processing.
- Conversion of audio data to 16,000 Hz sampling rate.
- Creation of a vocabulary from the cleaned text data.

Vocabulary Creation

A unique vocabulary was created by combining unique characters from both the training and test datasets, including special tokens '[UNK]' and '[PAD]'.

Data Preprocessing and Transformation

The preprocessing function included in the model setup transformed audio data into input values, and sentences were encoded into token IDs using the tokenizer and feature extractor. Padding and attention mask handling were managed to ensure proper training.

Model Training and Evaluation

Model Setup

The model used was ‘facebook/wav2vec2-large-xlsr-53‘. The following hyperparameters were configured:

- Attention Dropout: 0.1
- Hidden Dropout: 0.1
- Feature Projection Dropout: 0.0
- Mask Time Probability: 0.05
- Layer Dropout: 0.1
- CTC Loss Reduction: Mean
- Pad Token ID: 157

The feature extractor and tokenizer were customized for Arabic, and a DataCollator was defined for dynamic padding.

Training Arguments

Training was conducted over 5 epochs using the following parameters:

- Batch Size: 8
- Gradient Accumulation Steps: 8
- Learning Rate: 1e-4
- Warmup Steps: 500
- Evaluation Strategy: Steps (every 400 steps)
- FP16 Training: Enabled

Training Process

The model was trained using the Trainer API, with evaluation occurring after every 400 steps. The trainer used the defined data collator to handle padding and masking correctly. The computation of the Word Error Rate (WER) metric was used for evaluating the model.

Evaluation Metrics

The WER was computed for both training and test sets. For each step during evaluation, the computed WER was logged, and the best models were saved.

Results

- **Final Training WER**: **0.500657**
- **Final Test WER**: **0.491042**

Model Saving

The trained model and processor were saved at the specified directories:

- **Trained Model Path**: **'./wav2vec2-arabic-model'**
- **Processor Path**: **'./wav2vec2-arabic-processor'**

Conclusion

This report details the process from dataset collection, preprocessing, model training, and evaluation to the final model saving. The metrics achieved reflect the model's performance on Arabic speech recognition tasks.

Training Results

Step	Training Loss	Validation Loss	WER
400	9.627800	3.030787	1.000000
800	2.236600	0.663298	0.768855
1200	0.716400	0.419083	0.588736
1600	0.539000	0.358852	0.537523
2000	0.475500	0.336499	0.512864
2400	0.438100	0.317160	0.500657
2800	0.411500	0.314824	0.491042

Table 1: Training and Validation Results with WER

Egyptian Arabic Dataset: Collected by scraping audio data from sources such as YouTube, podcasts, and Egyptian TV programs.

Code Explanation

The Python code is designed to handle audio processing and transcription from YouTube playlists. Below is the step-by-step explanation of the code.

Code Breakdown

- **Imports**:
  - **Libraries**:
    - `os`: Provides a way of using operating system-dependent functionality like reading or writing to the file system.
    - `gc`: Implements automatic garbage collection to free up memory.
    - `speech_recognition`: A library for performing speech recognition, allowing audio input to be converted to text.
    - `pydub`: Facilitates audio processing, including manipulation and conversion of audio files.
    - `yt_dlp`: A fork of youtube-dl that allows downloading videos from YouTube and other sites.
    - `pandas`: A powerful data manipulation and analysis library, particularly useful for handling structured data.
    - `spleeter`: A library for separating audio sources, commonly used in music processing.
- **Mount Google Drive**:
  - The code mounts Google Drive to Google Colab for saving processed files and transcription outputs.
- **Functions**:
  - `clear_memory()`: Clears memory by performing garbage collection.
  - `remove_silence(audio)`: Removes silence from audio using the pydub library's silence detection functionality.
  - `process_audio_chunk(audio_chunk, chunk_index, video_id)`: Processes individual chunks of audio, saves them, transcribes them, and returns transcriptions.
  - `process_video(video_url)`: Handles video processing by downloading the audio, cleaning it, chunking, and transcribing. Results are saved in a CSV file.
  - `extract_video_urls_from_playlist(playlist_url)`: Extracts video URLs from a YouTube playlist.
  - `process_playlist(playlist_url)`: Processes all videos from a playlist, handles transcriptions, and saves results.

article amsmath

Usage:

- We created a playlist on YouTube and collected some videos to scrape them.

```
$playlisturl = 'https://www.youtube.com/playlist?list=
PLwR8LGk84674ckfo6BGR0f5ioSqzB55_A'
$process_playlist(playlisturl)
```

Explanation of the Code

This section outlines the steps taken for processing audio data and training a model using an Egyptian Arabic dataset.

- **Text Preprocessing** The function `remove_special_characters` cleans up text transcripts by:
  - Removing special characters, Arabic diacritics, and unnecessary spaces.
  - Normalizing Arabic characters replacing variations of with a standard form.
  - Ensuring only Arabic characters are retained.
- **Vocabulary Preparation** The `build_vocab` function:
  - Reads transcripts from multiple CSV files.
  - Processes them using `remove_special_characters`.
  - Creates a character-level vocabulary from unique characters in the cleaned text.
  - Adds special tokens like [UNK] (unknown), [PAD] (padding), and maps space to —.
  - Saves the vocabulary as a JSON file for tokenizer initialization.
- **Processor Initialization** The `initialize_processor` function:
  - Initializes a `Wav2Vec2CTCTokenizer` using the created vocabulary.
  - Sets up a `Wav2Vec2FeatureExtractor` for audio preprocessing (resampling, padding, and normalization).
  - Combines these into a `Wav2Vec2Processor` for managing input preparation.
  - Saves the processor for reuse.
- **Dataset Class** The `AudioDataset` class:
  - Handles on-the-fly audio loading, resampling, and transcript tokenization.
  - Converts audio files into numerical input values and transcripts into tokenized labels.
  - Ensures files and transcripts exist and handles edge cases like missing extensions.
- **Data Collator** The `DataCollatorCTCWithPadding` class:
  - Pads audio inputs and tokenized labels to ensure uniform batch sizes.
  - Masks padding in the labels with -100 to ignore during loss calculation.
- **Datasets and DataLoaders** Training and validation datasets are created using the cleaned CSV files and the `AudioDataset` class. Data loaders with the custom data collator prepare batches of data for the model.
- **Model Initialization** The `Wav2Vec2ForCTC` model:
  - Is initialized with pretrained Wav2Vec2 checkpoint facebook/wav2vec2-large-xlsr-53
  - Fine-tunes the model by adding a classification head with the vocabulary size.
  - Freezes the feature extractor layers to focus training on the CTC head.
- **Training Loop** The script trains the model for 2 epochs:
  - Training Phase: For each batch, predictions (outputs) are computed, and loss (outputs.loss) is calculated. Model parameters are updated using backpropagation.
  - Validation Phase: Gradient calculation is disabled for efficiency. Predictions and loss for the validation set are computed. Predictions and references are decoded for evaluation.

- **Metrics** Two metrics are calculated:
  - WER (Word Error Rate): Proportion of incorrect words compared to the total number of words in references.
  - CER (Character Error Rate): Proportion of incorrect characters compared to the total number of characters in references.
  - The `calculate_cer` function manually computes CER by comparing predicted and reference transcripts.
- **Training Outputs** For each epoch, the script prints:
  - Average training loss.
  - Validation loss.
  - WER and CER for validation.

Model Saving

The trained model and processor were saved at the specified directories:

- **Trained Model Path**: **'./wav2vec2-arabic-model2'**
- **Processor Path**: **'./wav2vec2-arabic-processor2'**

Training Results for Egyptian Arabic Dataset

The model trained on the Egyptian Arabic dataset produced the following results:

- **Epoch 1:**
  - Loss: 2.283299792775672
  - Validation WER: 0.7395340026520174
  - Validation CER: 0.0244
- **Epoch 2:**
  - Loss: 0.8704423548760818
  - Validation WER: 0.6258761129001705
  - Validation CER: 0.0315
- **Epoch 3:**
  - Loss: 0.6984904283136588
  - Validation WER: 0.5748247774199658
  - Validation CER: 0.0358