Below is a detailed report on building a basic speech-to-text system using pre-trained models and libraries such as `SpeechRecognition` and `Wav2Vec`. This report includes an overview of the components, installation instructions, code implementation, and testing. --- # Report on Building a Basic Speech-to-Text System ## Introduction Speech recognition technology enables the conversion of spoken language into text. This report outlines the steps to create a basic speech-to-text system using Python libraries, specifically `SpeechRecognition` and `Wav2Vec`. The system will be capable of transcribing short audio clips. ## Components 1. **Python**: The programming language used for implementation. 2. **SpeechRecognition Library**: A library that provides easy access to various speech recognition engines and APIs. 3. **Wav2Vec**: A pre-trained model for automatic speech recognition (ASR) developed by Facebook AI Research. 4. **Audio Files**: Short audio clips in formats like WAV or MP3 for testing. ## Installation To set up the environment, you need to install the required libraries. You can do this using `pip`. Open your terminal or command prompt and run the following commands: ```bash pip install SpeechRecognition pip install pydub pip install torch torchvision torchaudio pip install transformers ``` ### Additional Dependencies - **pydub**: This library is used for audio file manipulation. - **torch**: Required for running the Wav2Vec model. - **transformers**: A library by Hugging Face that provides pre-trained models. ## Implementation ### Step 1: Import Libraries ```python import speech_recognition as sr from pydub import AudioSegment import torch from transformers import Wav2Vec2ForCTC, Wav2Vec2Tokenizer ``` ### Step 2: Load the Wav2Vec Model ```python # Load the pre-trained Wav2Vec2 model and tokenizer tokenizer = Wav2Vec2Tokenizer.from_pretrained(“facebook/wav2vec2-base-960h”) model = Wav2Vec2ForCTC.from_pretrained(“facebook/wav2vec2-base-960h”) ``` ### Step 3: Function to Transcribe Audio ```python def transcribe_audio(file_path): # Load audio file audio = AudioSegment.from_file(file_path) # Convert audio to mono and set the sample rate to 16kHz audio = audio.set_channels(1).set_frame_rate(16000) # Export audio to a temporary WAV file temp_file = “temp.wav” audio.export(temp_file, format=”wav”) # Load the audio file for transcription speech, _ = torchaudio.load(temp_file) input_values = tokenizer(speech.squeeze().numpy(), return_tensors=”pt”).input_values # Perform inference with torch.no_grad(): logits = model(input_values).logits # Get predicted ids predicted_ids = torch.argmax(logits, dim=-1) # Decode the ids to text transcription = tokenizer.batch_decode(predicted_ids)[0] return transcription ``` ### Step 4: Main Function to Run the System ```python def main(): # Specify the path to your audio file audio_file_path = “path/to/your/audio/file.wav” # Transcribe the audio transcription = transcribe_audio(audio_file_path) # Print the transcription print(“Transcription:”, transcription) if __name__ == “__main__”: main() ``` ## Testing 1. **Prepare an Audio File**: Ensure you have a short audio clip in WAV format. You can convert MP3 files to WAV

using tools like `pydub`. 2. **Run the Script**: Execute the Python script. The transcription of the audio file will be printed in the console. ## Conclusion This report outlines the steps to create a basic speech-to-text system using Python libraries. By leveraging pre-trained models like Wav2Vec, we can achieve accurate transcriptions of short audio clips. This system can be further enhanced by integrating more advanced features such as noise reduction, handling different languages, and improving the user interface. ## Future Work - **User Interface**: Develop a graphical user interface (GUI) for easier interaction. - **Language Support**: Extend the system to support multiple languages. - **Real-time Transcription**: Implement real-time speech recognition capabilities. --- This report provides a comprehensive guide to building a basic speech-to-text system.

Building a basic speech-to-text system using pre-trained models is a manageable task with the help of libraries like `SpeechRecognition` for traditional approaches and `wav2vec` for more advanced neural network-based methods. Below, I will outline the steps needed to create a simple system capable of transcribing short audio clips.

Step 1: Setup Your Environment

1. **Install Required Libraries:**

Make sure you have Python installed (preferably Python 3.6 or higher). You need to install the following packages:

```bash
Pip install SpeechRecognition pyaudio
```

For the `wav2vec` model, you may want to use the `transformers` library:

```bash
Pip install transformers torch torchvision torchaudio
```

Step 2: Transcription Using SpeechRecognition

Here's a basic implementation using the `SpeechRecognition` library, which may rely on online services such as Google Web Speech API.

Code Implementation:

```python
Import speech_recognition as sr

Def transcribe_audio(audio_file):
  # Initialize recognizer
  Recognizer = sr.Recognizer()

  # Open the audio file and recognize the speech
  With sr.AudioFile(audio_file) as source:
    Audio_data = recognizer.record(source)  # read the entire audio file

  Try:
    # Recognize speech using Google Web Speech API
    Text = recognizer.recognize_google(audio_data)
    Print("Transcription: ", text)
  Except sr.RequestError as e:
    Print("Could not request results from Google Speech Recognition service; {0}".format€)
  Except sr.UnknownValueError:
```

Print(“Google Speech Recognition could not understand audio”)

If __name__ == “__main__”:

Audio_file_path = ‘path_to_your_audio_file.wav’ # Replace with your actual file path

Transcribe_audio(audio_file_path)

```

### Step 3: Transcription Using Wav2Vec

If you’re interested in a more advanced transcription using the `wav2vec` model, please follow these steps:

#### Code Implementation:

```python

Import torch

From transformers import Wav2Vec2ForCTC, Wav2Vec2Tokenizer

Import torchaudio

Def transcribe_audio_with_wav2vec(audio_file):

# Load pretrained model and tokenizer

Tokenizer = Wav2Vec2Tokenizer.from_pretrained(“facebook/wav2vec2-base-960h”)

Model = Wav2Vec2ForCTC.from_pretrained(“facebook/wav2vec2-base-960h”)

# Load audio file

Waveform, sample_rate = torchaudio.load(audio_file)

```
    # Resample the audio if needed
    If sample_rate != 16000:
        Resampler = torchaudio.transforms.Resample(orig_freq=sample_rate, new_freq=16000)
        Waveform = resampler(waveform)

    # Get transcription
    Input_values = tokenizer(waveform.squeeze().numpy(), return_tensors='pt', padding='longest').input_values

    With torch.no_grad():
        Logits = model(input_values).logits

    Predicted_ids = torch.argmax(logits, dim=-1)

    # Decode the ids to get the final transcription
    Transcription = tokenizer.batch_decode(predicted_ids)

    Print("Transcription: ", transcription[0])

If __name__ == "__main__":
    Audio_file_path = 'path_to_your_audio_file.wav' # Replace with your actual file path
    Transcribe_audio_with_wav2vec(audio_file_path)
```

Step 4: Use Your System

1. **Prepare an Audio File:**

Make sure you have a short audio file in WAV format with clear speech.

2. **Run the Script:**

Execute the script by calling the appropriate function depending on which model you have implemented (either `transcribe_audio` or `transcribe_audio_with_wav2vec`).

Note:

- Make sure the audio file is formatted correctly (16kHz, mono) for the wav2vec model.

- The `SpeechRecognition` library can use various APIs, while `wav2vec` is an offline method as it processes the model locally.

- Check API usage limits and documentation for the chosen libraries if you use web-based solutions.

With these steps, you should have a functional speech-to-text system capable of transcribing short audio clips!

We have others sources that required and present in browser s with full report.